DOCUMENT RESUME

ED 082 487 EM 011 456

AUTHOR	Mudge, J. C.
TITLE	Human Factors in the Design of a Computer-Assisted Instruction System. Technical Progress Report.
INSTITUTION	North Carolina Univ., Chapel Hill. Dept. of Computer Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	UNC-TPR-CAI-7
PUB DATE	Jun 73
NOTE	324p.; Thesis submitted to the Department of Computer Science, University of North Carolina
EDRS PRICE	MF-$0.65 HC-$13.16
DESCRIPTORS	*Computer Assisted Instruction; Computer Programs; Doctoral Theses; *Human Engineering; Interaction; *Man Machine Systems; Programers; Programing Languages; *Systems Development; Technical Reports
IDENTIFIERS	CAI; DIAL; DIAL 2; *Display Based Interactive Author Language; Sieve; Translator Writing System; TWS

ABSTRACT

A research project built an author-controlled computer-assisted instruction (CAI) system to study ease-of-use factors in student-system, author-system, and programer-system interfaces. Interfaces were designed and observed in use and systematically revised. Development of course material by authors, use by students, and administrative tasks were integrated into one system whose nucleus was a display-based interactive author language (DIAL). The design permitted systematic language implementation and easy language modification and used a translator writing system (TWS) to generate compilers. Authoring by teachers required simplicity of the language and its operational environment. A measured high level of user acceptance proved the design to be sound, and a significant reduction in authoring time was achieved. DIAL was observed to be a superior language, for machine intrusion was low and other syntactic improvements were possible. An answer-evaluating technique, called the sieve, was devised and a syntactically improved DIAL/2 language derived. The TWS helped to implement DIAL and to remediate language weaknesses. Although the TWS was not available for the command language of the operational environment, the human-factors debugging period revealed the desirability of such. (Author/PB)

FILMED FROM BEST AVAILABLE COPY

ED 082487

University of North Carolina at Chapel Hill

Department of Computer Science

EM 011456

ED 082487

HUMAN FACTORS IN THE DESIGN OF A COMPUTER-ASSISTED INSTRUCTION SYSTEM

J.C. Mudge

June 1973

Technical Progress Report CAI-7
to the
National Science Foundation

under Grant GJ-755

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

DEPARTMENT OF COMPUTER SCIENCE

University of North Carolina at Chapel Hill

HUMAN FACTORS IN THE DESIGN OF A COMPUTER-ASSISTED INSTRUCTION SYSTEM

by

Jonathon Craig Mudge

A Dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

1973

Approved by:

Fred'k P. Brooks Jr
Adviser

Peter Calingaert
Reader

Gyula Magó
Reader

JONATHON CRAIG MUDGE. Human Factors in the Design of a Computer-Assisted Instruction System (Under the direction of DR. FREDERICK P. BROOKS, JR.)

This research is an exploratory case study of ease-of-use factors in man-computer interfaces.

The approach has been to build and evaluate a real man-machine system, giving special attention to the form of all man-machine and machine-man communications. Author-controlled computer-assisted instruction was selected. Such a system has three principal interfaces: student-system, author-system, and computer programmer-system.

The method was to design, build a large subset of the design, make systematic observations of the three interfaces in use, and then iterate on the design and on the observations. The system has been in regular class use for a year.

Interactive development of course material by authors, execution of instructional programs by students, and the requisite administrative tasks are integrated into a single production-oriented system. The nucleus of the system is a new display-based interactive author language, DIAL. The design demands a language implementation which is systematic and which permits easy language modification. A translator writing system, based extensively on McKeeman's, assists computer programmers in generating compilers for new versions of the language.

Two of the design assumptions (that the course author is always an experienced teacher and that he does his own programming in DIAL, without an intermediary CAI language programmer) are major departures from most CAI authoring systems. Professorial-level authoring imposes stringent requirements on the ease-of-use and simplicity of the language and the operational environment in which it is embedded.

A measured high level of user acceptance proved the soundness of the design and illuminated the relatively few mistakes made. A factor-of-five improvement in authoring time over data published for other systems was observed. Several improvements in DIAL over existing CAI languages were observed. The underlying machine intrudes much less than in existing languages, and there are other syntactic improvements. The provision in DIAL of a pattern matching function allowed a very general answer-evaluating technique, called the sieve, to be devised. Analysis of author use of DIAL has derived DIAL/2, which is radically different syntactically but only slightly enriched functionally.

The translator writing system proved very useful in progressive implementation of DIAL and in the remediation of language weaknesses as they were discovered. Although a translator writing system was not available for the command language of the operational environment, the human-factors debugging period (necessary for all user-oriented systems) revealed the desirability of such.

To my parents

ACKNOWLEDGMENTS

I am greatly indebted to Professor Frederick P. Brooks, Jr., for his insight, for many valuable suggestions, and for his guidance throughout this research. For his tutorials on machine design and for his authoring use of my system, I am also most grateful.

I thank O. Jack Barrier for his help in the long days and nights of the CAI System's first use in production and for subsequently assuming responsibility for the continued operation and improvement of the system. I also thank Gary D. Schultz for his ultra-reliable CHAT System, William H. Blair for the support provided by his deep systems programming knowledge, and the proctors who manned the system during the Fall and Spring class use.

Professor Peter Calingaert's encouragement and interest are gratefully acknowledged.

My wife, Anne, provided a great deal of encouragement and understanding.

I am grateful to the National Science Foundation for partial support under NSF Grant Number GJ-755 and for the hardware facility, provided under the University Science Development Program.

CONTENTS

Chapter Page

1. INTRODUCTION 1
   - 1.1 User-orientation in system design 1
   - 1.2 An approach to research in computer-assisted instruction (CAI) 5
   - 1.3 The UNC CAI Project 7
   - 1.4 Language design in computer science 9
2. EXISTING CAI SYSTEMS AND LANGUAGES 11
   - 2.1 The scope of the chapter 11
   - 2.2 Coursewriter systems 14
   - 2.3 Other languages 23
   - 2.4 Discussion 32
3. THE DESIGN PHILOSOPHY UNDERLYING DIAL 37
   - 3.1 Motivation 37
   - 3.2 *A priori* design decisions 39
   - 3.3 Objectives for the language *per se* 43
   - 3.4 Objectives for the interactive programming system 45
   - 3.5 Why not adapt a general purpose programming language? 47
4. THE DIAL LANGUAGE 53
   - 4.1 Introduction 53
   - 4.2 Writing a simple instructional program 55
   - 4.3 Default branching in a program 64
   - 4.4 Classifying recognized responses 67
   - 4.5 Expressions and assignment statements 68
   - 4.6 The IF-statement for branching 70
   - 4.7 Non-exact matching 74
   - 4.8 The sieve 78
   - 4.9 The naming-statement 81

4.10 Repetition constructs 82
4.11 Cathode-ray tube screen formatting 84
4.12 Light pen usage 89
4.13 The RESUME-statement 92
4.14 Subroutines 92
4.15 Input-output synchronization 95
4.16 DIAL specifications 99

5. THE OPERATIONAL ENVIRONMENT 124

5.1 The host computer system 124
5.2 The Chapel Hill Alphanumeric Terminal (CHAT) System 128
5.3 The student/author work station 130
5.4 System overview 134
5.5 Instructional programming in DIAL - author use of the system 138
5.6 The execution of an instructional program - student use of the system 150
5.7 Proctor facilities 154

6. MODIFYING AND EXTENDING DIAL - THE TRANSLATOR WRITING SYSTEM 157

6.1 Introduction 157
6.2 The compiler and the CAI System 159
6.3 The CAI translator writing system 162
6.4 Steps in using the CAI translator writing system 171
6.5 The class of grammars acceptable to the translator writing system 173

7. EXPERIMENTAL METHOD AND RESULTS 175

7.1 Introduction 175
7.2 Collection of student use data 176
7.3 Analysis of student use data 185
7.4 Author use data 191

8. DISCUSSION 194

8.1 Introduction 194
8.2 The student-system interface 195
8.3 The author-system interface 201
8.4 The computer programmer - system interface 237
8.5 Observations about human factors 240
8.6 The cost of designing and implementing the system 244
8.7 Is it widely applicable? 247

9. SUGGESTIONS FOR FURTHER WORK 251

9.1 Introduction 251
9.2 DIAL/2 252
9.3 Author-defined commands for student use 259
9.4 Debugging aids for DIAL programming 263
9.5 Answer processing subsystems 264
9.6 A man-machine interface for unrecognized answers 266
9.7 More service programs 268
9.8 Color cathode-ray tube terminals 270

APPENDIX

A Questionnaire A and summaries of student responses 273
B Questionnaire B and summaries of student responses 280
C Posttest 304
D Program listing of the CAI System (bound separately)

SELECTED BIBLIOGRAPHY 306

LIST OF FIGURES

Figure — Page

A student/author work station showing cathode-ray tube, keyboard, light pen, slide projector and screen — Frontispiece

4.1 The metalanguage used to define the syntax of DIAL — 58

4.2 A DIAL program segment showing a sieve for answer evaluation of the expected response SAM:DO X=1 TO 50; — 79

4.3 The slide struc4 used in Example 4.6 — 91

4.4 An execution of Example 4.6 — 91

4.5 A DIAL machine — 101

4.6 The "onion structure" implementation of the CAI System — 103

5.1 The CHAT System hardware configuration — 129

5.2 The CHAT Monitor Table of Contents (MTOC) display — 131

5.3 An overall flowchart of the CAI System — 135

5.4 The cathode-ray tube screen format in author mode — 141

5.5 The summary sheet of commands for work station use — 147

5.6 The cathode-ray tube and slide displays during a typical student session — 151

5.7 The logic for setting the RECOVNEEDED switch — 153

Figure Page

6.1 The two parts of the translator writing system 160

6.2 The invocation of COMPILER by the controlling routine AUTHOR 161

6.3 The main compilation loop in COMPILER showing the relationship between ANALYZE, SCAN and CODEGEN 165

6.4 A lexical flowgraph for an early version of DIAL 170

6.5 The CODEGEN sections corresponding to three productions in a recent version of DIAL 172

8.1 A typical sequence of author sessions 207

8.2 A change being made to a statement 210

CHAPTER 1

INTRODUCTION

1.1 User-orientation in system design

1.1.1 Users, by and large, do not feel that computer systems are yet accommodatingly matched to their human users. People are flexible and can make remarkable adaptations to machine inflexibilities. Machine designers have always exploited that flexibility, sometimes ruthlessly. However, the more a user is forced to adapt, the less productive he will be. Thus, just as aircraft-cockpit designers have done, computer system designers are turning to the study of ease-of-use factors in the man-machine interface.

1.1.2 The approach in this thesis research has been to design and evaluate a real man-machine system. Author-controlled computer-assisted instruction was selected for three reasons.

(1) The man-machine system required was small enough for one person to design a total, production-oriented, hence real, system.

(2) The resulting system was potentially useful for instruction at the University.

(3) The closest possible user feedback obtains; the tool builders and users are one.

The method of investigation was to design, build a large subset of the design, make systematic observations of users, and then iterate on the design and on the observations.

The nucleus of the design is a display-based interactive author language, DIAL. Interactive development of course material by authors, execution of instructional programs by students, and the requisite administrative tasks are integrated into a single system. The design demands a language implementation which is systematic and which permits easy language modification. A translator writing system, based extensively on McKeeman's, assists computer programmers in generating compilers for new versions of the language.

The evaluation of the design focusses on the three principal interfaces: student-system, author-system, and computer programmer-system.

Two of the design assumptions (that the course author is always an experienced teacher and that he does his own programming in DIAL, without an intermediary CAI language programmer) are major departures from most CAI authoring systems. Professorial-level authoring imposes stringent requirements on the ease-of-use and simplicity of the language and the operational environment in which it is embedded. Because it has been forced to adapt more to the user than most CAI systems, this system is probably a better tool for studying human factors than most.

1.1.3 The scope of the research is the syntactic as opposed to the semantic elements of a man-machine system. That is, I am concerned with the form of communication between a system and its user, as contrasted with its meaning. The more obvious syntactic elements include programming language syntax and command-language syntax. I also view the following as essentially syntactic in the sense of pertaining to form of communication: the physical work-station design and its operating procedures; the removal of redundant operations in order to minimize user action; and the invention of general purpose primitives to subsume several special operations.

1.1.4 The following is a selection of the results of this research.

A measured high level of user acceptance proved the soundness of the design and illuminated the relatively few mistakes made. A factor-of-five improvement in authoring time over data published for other systems was observed.

Several improvements in DIAL over existing CAI languages were observed: there is much less intrusion of the underlying machine; some general, powerful, and easily used mechanisms have been borrowed from general programming languages; and a consistent syntax embodies its functional facilities.

Analysis of author use of DIAL has derived DIAL/2, which is radically different syntactically but only slightly enriched functionally. This contrasts with my early prediction that DIAL would be deficient semantically, not syntactically.

The translator writing system proved very useful in progressive implementation of DIAL and in the remediation of language weaknesses as they were discovered. It is expected to be very useful in implementing DIAL/2.

Although a translator writing system was not available for the command language of the operational environment, the human-factors debugging period (inherent in user-oriented systems) revealed the desirability of such.

The provision in DIAL of a pattern matching function allowed a very general answer evaluating technique, called the sieve, to be devised.

1.2 An approach to research in computer-assisted instruction (CAI)

The last decade has seen computer-assisted instruction go through two stages, an experience not uncommon in several computer application areas. In the first stage, potential users were led to expect great benefits from the use of computers in the instructional process. The rapid increase in the availability of time-sharing systems added support to the arguments of the CAI proponents. The second stage has been the realization that the success of CAI has fallen far short of the claims made for it, and that a reassessment of approaches should be made.

The approach taken in this thesis postulates that real measurable progress can be made by restricting the instructional subject matter to subjects which are highly structured. Elementary computer programming has been chosen from the class of structured subjects for two reasons: (1) researchers will be professionally familiar with the subject matter and its pedagogy, and (2) the research results can be directly applied and measured in an existing course at the University.

It must be emphasized that this project is concerned with computer science aspects of CAI rather than purely educational aspects. Examples of efforts in this latter category are those by Stolurow [1968], who is concerned with the psychological foundations of teaching and learning, and by Bunderson [1970], whose major research effort is on the design of instructional programs. Moreover, there are computer science areas which have been excluded from the scope of this project, e.g., CAI hardware research (being attacked on a large scale by the PLATO project [Bitzer and Skaperdas, 1970]), and 'natural' language question-answering research. This latter area is typical of many research areas which have long-term applications in CAI, and relate to the system designed in this project insofar as its flexibility allows new developments to be incorporated as they become available.

Two papers which do well in capturing the current status of CAI are [Bundy, 1968] and [Alpert and Bitzer, 1970].

To be a valid tool, both for the University and for a study of human factors, the system designed has to be total and production-oriented. To be total, it has to serve all classes of users associated with the CAI environment: students, authors, proctors, and instructors in charge of CAI classes.[1] The implications of a production, rather than an experimental, system permeate both design and implementation. The production implications include:

---

[1] In this thesis, this operational environment is collectively referred to as the CAI System, or, for short, the System.

(1) concurrent multiple-author and multiple-student use;

(2) complete treatment of abnormal termination (ABEND) situations;

(3) comprehensive supporting programs for administrative jobs;

(4) procedures for recovery after system failure;

(5) efficient programming of the implementation;

(6) operational procedures for the CAI Center.

1.3 The UNC CAI Project

The current computer-assisted instruction work at the University falls into several project areas, which may roughly be classified into student-controlled CAI, teacher-controlled CAI, and author-controlled CAI. This thesis results from work done in Phase II of the author-controlled project. Phase I was the development of a conventional Computer Administered Programmed Instruction (CAPI) System [Brooks, 1970] using an IBM 1050 audio-visual terminal (typewriter-based), for teaching a beginning course in the programming language PL/I.

The Phase I system was converted, with as few changes as possible, to operate on a Computer Communications Inc.

CC-30 terminal (CRT-based). As a result, feasibility of this terminal for the envisaged CAI work was established and the Phase II objectives laid down.

This second phase called for the following.

(1) the installation of new communications equipment: a medium-speed leased line to the Triangle Universities Computation Center (TUCC), a general purpose campus computing facility, to give character speeds beyond the 15 character/second possible via dial-up facilities.

(2) six CC-30 terminals, each consisting of a CRT, keyboard, light pen, and random-access slide projector.

(3) the building of a monitor program for the CAI region of Large Capacity Storage at TUCC to control the programs in the region and to provide communications device programming support.

(4) the design and building of student/author work stations based on the CC-30 terminals.

(5) the development of a CAI control program which simultaneously presents course material from several different courses to several student work stations.

(6) the design of an author language and building of an incremental compiler which generates code for presentation as course material by the CAI control program.

(7) the preparation of course material for an elementary programming course in PL/I.

(8) evaluation of the system by class trial.

Mr. Gary D. Schultz [1973] reports items (1), (2), and (3), which form his Chapel Hill Alphanumeric Terminal (CHAT) System.

This thesis reports items (4) through (8). The course material, item (7), was programmed by Professor F. P. Brooks, Jr. The off-line supporting programs, e.g., the student file maintenance program, were written by other CAI Project workers (Section 5.7) to my external specifications. My contributions are the design of (4), (5), (6), and (8), and the implementation of (5), (6), and (8).

1.4 Language design in computer science

Language design is an art. Given a particular objective, there is no algorithm for designing a programming language to meet that objective; there is not even any clear delineation of various tradeoffs involved, let alone any quantitative measures of them [Sammet, 1969]. In the remainder of the thesis, I therefore:

attempt to define the objectives which have been set;

discuss the design decisions made (pointing out what are thought to be tradeoffs involved);

attempt to see if the objectives have been met.

CHAPTER 2

EXISTING CAI SYSTEMS AND LANGUAGES

2.1 The scope of the chapter

A wide variety of languages and systems is being used for programming interactive use of computers for instructional purposes.

Before turning to describe that subset which is appropriate for this chapter, we must define the area of instructional use of computers at which DIAL is directed.

This area is based on presentation of successive "frames" or units of course material, to a student. In each frame the student is expected to respond in an anticipated manner to the material presented to him in that frame. The order of presentation of frames is under the control of an author by means of the instructional strategy he has programmed into his course material. Since an author is the agent controlling the interaction between student and computer system, this area is usually called author-controlled CAI. Two other modes can be distinguished. Consider a computer graphics system programmed to display step-by-step graphic solutions to numerical analysis problems [Oliver and Brooks, 1969]. Hands-on laboratory use of this system by individual students is an example of student-controlled CAI. Teacher-controlled CAI is exemplified by the use of this same system by a teacher in front of a class of students.

In this chapter, then, I discuss specially devised author languages for author-controlled CAI; IBM's Coursewriter typifies this class of languages.

The chapter specifically excludes the following.

1. Discovery-mode systems

Here the student presents questions, usually phrased in natural language, to a system which interrogates a highly-structured data base specially prepared to cover the subject matter being taught. Examples of such systems are Thompson's REL System [1969], Simmons's PROTOSYNTHEX III [1970] and Carbonell's SCHOLAR [1970]. Incidentally, experience with these experimental systems is aiding research being done on the linguistic and semantic analysis of constructed responses in author-controlled CAI.

2. Conversation machines

ELIZA [Weizenbaum, 1966] is the best known of those systems which, in providing conversation within a limited context, have been used for CAI.

3. Interactive programming languages

Some writers include in CAI the use of interactive programming languages by students for working exercises, programming simulations, etc. This aspect of CAI is clearly excluded from this chapter; however, such languages have been adapted as author languages, and these are discussed in Chapter 3.

4. Highly tailored systems

An interesting example of this type of system is the TEACH System [Fenichel, _et al_., 1970] developed at the Massachusetts Institute of Technology to ease the cost and improve the results of elementary instruction in computer programming. The system was designed for this subject matter only, and includes a specially designed programming language, UNCL, in which a student writes specific programs requested of him by TEACH. The system, by monitoring the execution of such a program, attempts to present meaningful feedback to the student.

5. Course generators

An author enters course material as a structured file; standard procedures draw exercises from the file and present them to a student. Such systems are particularly useful for drill-and-practice and have so been used for arithmetic and spelling drill. Examples of such systems are TSA (Teacher

Student Algol) at Stanford University [Suppes, *et al*., 1968] and CG-1 (Course Generator) [Meadow, *et al*., 1968].

Author languages for such systems are characterized by having

(1) data structures - usually vectors - for storing questions, answers and scores,

and (2) a greater naming freedom than the languages described in later sections of this chapter.

A recent system developed by Wexler at the University of Wisconsin [Wexler, 1970a and b] should generate course material and interactions which appear quite intelligent to the student. His system, which has a structured information net of factual material, has made use of results from artificial intelligence and natural language processing which are common to the discovery-mode systems mentioned above.

2.2 Coursewriter systems

2.2.1 The IBM Corporation is responsible for the most widely used author languages and CAI systems, all based on the original Coursewriter I for the IBM 1400 series of computers. Coursewriter II and Coursewriter III are available as program products from IBM and are functionally complete, serving students, authors, and administrative personnel in the CAI environment. Since CWIII is typewriter terminal based and CWII is CRT terminal based, they are not in a successor relationship nor are they strictly compatible, and features of both will be described.

2.2.2 Coursewriter III [IBM 1969a, b, and c; 1970a]

2.2.2.1 Hardware

The student/author station is an IBM 1050 typewriter terminal with slide projector. It uses a host computer system, the IBM System/360, via a low-speed communications line. A random-access audio tape can be attached to the station as a special option.

2.2.2.2 The CWIII Language

Variables operated on by a CW program are of six types and are described in the following table. They are called *storage areas*, and one copy is kept for each student.

Storage area	Name for addressing in CW language	Attributes of contents	
counters	c0 - c30	signed integer	
buffers	b0 - b6	character strings up to 100 characters in length	b0 is the student input buffer
return registers	r1 - r6	label	
switches	s0 - s31	binary digit	
course parameters	p0 - p31	binary digit	
auxiliary storage	a	byte string	

Other data types are character string constants and integer constants.

Operations (statements) have the syntactic form

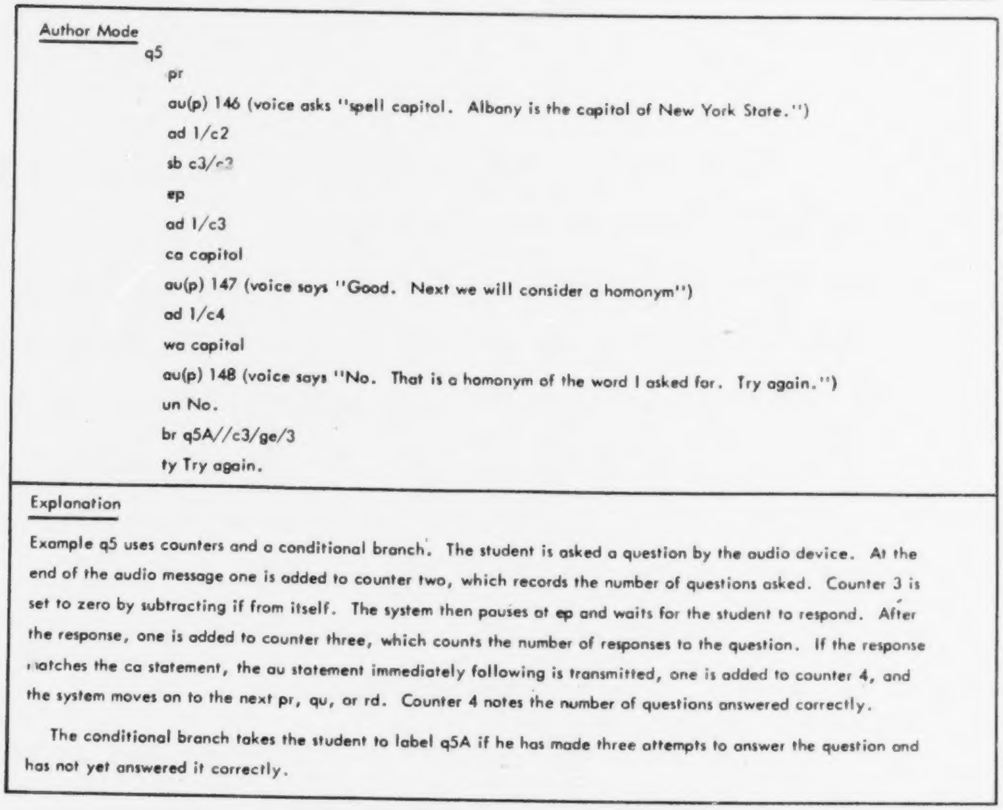
operation code	optional modifier	argument(s)

Text and questions are presented by the qu, rd, and ty operation codes which take as arguments a character string constant or a buffer. Arithmetic is specified by the

ad, sb, mp, and dv

operation codes which take two integer arguments each. Assignment is specified by the ld operation code which takes two arguments (of the same type). Conditional and unconditional branching is effected by the br operation code .

The following example [IBM, 1969a:20] illustrates each of these.



Author Mode

```
q5
   pr
   au(p) 146 (voice asks "spell capitol.  Albany is the capitol of New York State.")
   ad 1/c2
   sb c3/c3
   ep
   ad 1/c3
   ca capitol
   au(p) 147 (voice says "Good.  Next we will consider a homonym")
   ad 1/c4
   wa capital
   au(p) 148 (voice says "No.  That is a homonym of the word I asked for.  Try again.")
   un No.
   br q5A//c3/ge/3
   ty Try again.
```

Explanation

Example q5 uses counters and a conditional branch. The student is asked a question by the audio device. At the end of the audio message one is added to counter two, which records the number of questions asked. Counter 3 is set to zero by subtracting if from itself. The system then pauses at ep and waits for the student to respond. After the response, one is added to counter three, which counts the number of responses to the question. If the response matches the ca statement, the au statement immediately following is transmitted, one is added to counter 4, and the system moves on to the next pr, qu, or rd. Counter 4 notes the number of questions answered correctly.

The conditional branch takes the student to label q5A if he has made three attempts to answer the question and has not yet answered it correctly.

Answer matching is specified by the operation codes

ca, cb, wa, wb, aa, and ab,

and certain misspellings can be ignored by a selective character string match specification.

One level of indirect addressing of a storage area can be specified with some operation codes. For certain statements, modifiers can be placed in parenthesis after the operation code to specify a modified operation. For example

ca(l) re*d

uses the l (line processing) modifier to eliminate the third character from response comparison.

Two powerful features of CW are provided by the fn operation code and the macro facility. The argument field of a fn statement names a machine language subroutine and the arguments to be passed to it. The macro facility permits authors to write frequently-used course statement sequences in a skeleton form. Macro definition and invocation facilities are similar to those found in the macro facilities of assembler languages.

In summary, the language can be characterized as an assembler-level language. Naming of data and the amount available seem to be the major restrictions. Auxiliary storage is used to increase the amount of space available, but its addressing is not by name but by absolute location, e.g.,

ld a, 738, 46/b3

means that an area whose leftmost byte position is byte 738 and whose rightmost is 783 is loaded into buffer 3.

With such restricted format, low-level commands, explicit addressing and restrictions on sizes of data areas, the software implementation can be both fast and frugal in memory usage. Such implementation benefits are obtained at the cost of author convenience.

2.2.2.3 Student use of the system

The student signs on by typing the command sign on, his student number preceded by the designation s and the name of the course for which he is registered. Once he has gained access to the system he has three commands available

help
go to
sign off

The go to command allows him to invoke a particular section from a list of course sections provided by the author.

2.2.2.4 Author use of the system

An author signs on in the same manner as a student, but precedes his number by the letter a.

CW statements are entered in fixed format using the 1052 Printer-Keyboard. An author can view the execution of his course by signing on in student mode by prefixing s to his number. Labelled segments of the course can be executed while he is in author mode by using the go to command. He returns from such an execution by entering "author".

Other author commands are concerned with editing his program:

insert after
delete
replace
move

A listing can be obtained by the command type.

2.2.2.5 Supervision and monitor commands

The production orientation of CWIII systems is reflected in the comprehensiveness of the set of commands available to the system supervisors and monitors. The commands are for administrative tasks associated with student registration on the system and for changing system parameters [IBM, 1969c and 1970a].

2.2.3 Coursewriter II

2.2.3.1 Hardware

An IBM 1500 Instructional System consists of a number of student/author stations connected locally to a dedicated computer, either the IBM 1800 or 1130. Each station consists of a CRT display with light pen, keyboard and slide projector. Random-access audio tape is optional.

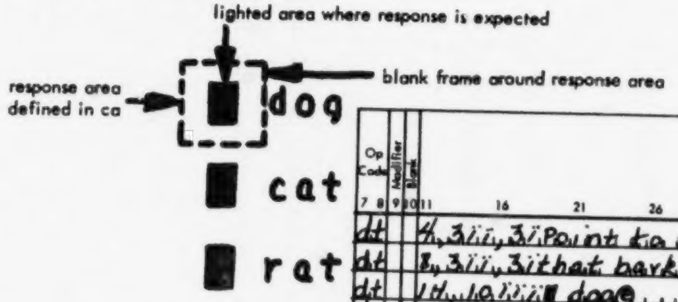
2.2.3.2 The CWII Language

The language is essentially the same as CWIII, with the differences arising from the properties of the terminal.

Explicit screen-formatting information is required. Light pen commands are included. In the following example [IBM, 1968: Part II, 77], the second and seventh statements show these two aspects. The former, the dt statement, displays "that barks" beginning at coordinates (8,3) on the CRT. The latter, the ca statement, specifies that a match occurs if the light pen touches the response area defined by a rectangle with its top left hand corner at coordinates (13,9) and depth 4 and width 3.

Point to the name of the animal

that barks.


lighted area where response is expected
blank frame around response area
response area defined in ca
dog
cat
rat

```
Op     Modifier Blank  Text
Code
7 8    9        10     11   16   21   26   31   36   41   46   51
dt                     4,3;;,3;Point to the name of the animal@
dt                     8,3;;,3;that barks.@
dt                     14,10;;;;■ dog@
dt                     18,10;;;;■ cat@          (note ■ is a special
dt                     22,10;;;;■ rat@           dictionary character)
ep     p               300@
ca     p               4,13,3,9;c1@
dt                     ;;;;yes, a dog barks.@
wa     p               4,17,3,9;w1@
dt                     ;;;A cat meows. Try again@
wa     p               4,21,3,9;w2@
dt                     ;;;A rat squeals. Try again@
```

A feature of CWII is the facility for special graphics display.

Indirect addressing as in CWIII is not provided.

2.2.3.3 Author use of the system

The author can operate in one of two modes: assembly or checkout mode. In assembly mode the editing and listing commands are as for CWIII. In checkout mode, he can view the execution of a course by entering the command EXECUTE. Provision is made for assembly from card input.

2.3 Other languages

2.3.1 WRITEACOURSE

This language was developed at the University of Washington by a project aimed at producing a language which is natural for the teacher, highly readable, and suitable for machine-independent implementation [Hunt and Zosel, 1968].

2.3.1.1 The language

The language, which is typewriter-terminal based, is very simple, having ten operations. The following example [Hunt and Zosel, 1968:926] illustrates display of text, use of counters (a number preceded by the symbol @), answer classification and branching, and its limited arithmetic capability.

```
(1)    SET @54,@41 TO 0 PRINT "WHAT DISCOVERY
(2)    LEAD TO LASERS?"|
(3)  3 ACCEPT CHECK "MASER" "QUASER"
(4)    "CANDLES" IF 1 CHECKS THEN GO TO 6|
(5)    ADD 1 TO @41 IF 0 CHECKS THEN GO TO 40|
(6)    IF 2 CHECKS THEN PRINT "THAT IS IN ASTRONOMY."
(7)    GO TO 40|
(8)    IF 3 CHECKS THEN PRINT "DO NOT BE SILLY."|
(9)40  IF @41 < 3 THEN PRINT "TRY AGAIN" GO TO 3|
(10)   ADD 1 TO @54 PRINT "THE ANSWER IS MASER."|
(11)6  PRINT "HERE IS THE NEXT QUESTION"|
```

In the example, the first statement initializes counters 54 and 41 and prints a question. The statements in lines (3) and (4) specify three anticipated responses. If the first response matches the student's answer ("IF 1 CHECKS") then the program branches to statement 6 on line (11). Unrecognized ("IF 0 CHECKS") and wrong answers cause incrementation of counter 41 and a branch to line (9). There a test on counter 41 determines whether the answer should be printed.

Statements are grouped into <u>lessons</u>, and lessons are grouped into courses. Since it is possible to activate one lesson from another in the same course, a subroutine facility exists. It appears that parameterization would have to be done via counters known to both the invoking and invoked lessons.

2.3.1.2 The operational environment

After a student has gained access to the system he types XEQ and then supplies the lesson name and course name when requested.

The system is placed in author mode by the command ///COMPILE. A new lesson is begun by ///PROGRAM NEW lesson-name/course-name. Each statement is checked for syntax errors as it is entered. The editing commands are ///ADD, ///DELETE and ///LIST.

The WRITEACOURSE language translator, written in PL/I, is an incremental compiler producing an internal form which is interpreted at run time. This internal form is the only representation of the source code kept and is decompiled when a source listing is requested.

2.3.2 FOIL

This language was developed at the University of Michigan by a project aimed at producing a language which is easy to use, has good computational capability and whose implementation is sufficiently flexible to allow language modification [Hesselbart, *et al.*, 1969].

2.3.2.1 The language

The language is for a student/author station consisting of a teletypewriter with optional slide projector.

It is a high-level language, as the following example [Hesselbart, 1968:94] illustrates

```
TY   WOULD YOU LIKE TO CONTINUE THE EXERCISE
ACCEPT
  IF 'NO,' GO TO FINISH
  IF 'YES,OK'
     NUM = NUM + 1
     GO TO NEXT
  GO BACK PLEASE ANSWER YES OR NO
```

Specification of an answer set is done as follows. A set of keywords to be treated as equivalent is written, separated by commas, between single quotes (as YES and OK in the above example). Exact matching is indicated by enclosing the anticipated responses in quotation marks. A digit following a list of keywords specifies the number of keywords which must match if other than one; a percentage match can also be specified.

The language is computationally powerful, allowing arithmetic expressions and vector data.

There is no subroutine facility in FOIL; however, FORTRAN subroutines can be invoked.

The syntax of the language, although restrictive, is clean; moreover, source listings of programs reveal a clear logical flow. A good deal of the computational and logical power of the underlying machine is available to the author in a natural way.

2.3.2.2 The operational environment

A student, once having signed on to the FOIL system, enters $SOURCE NAME to commence the course NAME.

An author begins the creation of a course by entering

```
$$RUN FOIL 6=*MSOURCE* 7=*MSINK* 8=coursename
                       9=qualifier
$$SOURCE  *MSOURCE*
```

Card input facilities are also available.

The implementation of FOIL is by an interpreter written in FORTRAN. The stated rationale for this is to enable an implementation (1) having few constraints on the syntax of the language, (2) permitting easy transfer to other time-sharing systems, and (3) allowing easy modification of the language.

To view the execution of a course, the author signs on as a student. Limited editing can be done while he is signed on as a student. Substantial revision is done by using a text editor unrelated to the FOIL system.

2.3.3 PLANIT

The language was developed at System Development Corporation and is claimed to be a multipurpose language for computer-human interaction, and simple enough to allow non-programmers to use it easily [Feingold, 1967].

The student/author station is a teletypewriter connected to a time-sharing system. The PLANIT system operates in four modes: lesson-building, editing, execution, and calculation. The student has access to the last two modes, an author to all four.

A lesson is composed of a set of frames, of which there are five types: Problem, Question, Multiple Choice, Decision, and Copy.

Lesson-building is done interactively with the author entering data (questions and answers) into the fixed format of the frames. The following example [Feingold, 1967:549] illustrates this. Note that data entered by the author follow an asterisk typed out by the system.

System prompt and author response:	Comments:
*Q	the question frame
FRAME 2.00LABEL=*MATH	labelled MATH
2. SQ.	specify question
*LETS SEE WHAT YOU REMEMBER ABOUT TEMPERATURE. USING F FOR DEGREES *FAHRENHEIT AND C FOR DEGREES CENTIGRADE, WRITE THE FORMULA FOR *CONVERTING FROM DEGREES FAHRENHEIT TO DEGREES CENTIGRADE. *HINT: F=9*C/5+32 CONVERTS FROM CENTIGRADE TO FAHRENHEIT.	question (one line at a time)
*	end of question
3. SA.	specify answer
*0FORMULAS ON	turn on algebraic matching
A+C=(5/9)(F-32)	+ signifies correct answer, answer A
*B F=9*C/5+32	answer B
*C C=(5/9)*F-32	answer C
*	

```
4.  SAT.                             |specify action to be taken
*A F: B:7                            |A first time:Feedback &
                                     |          branch to frame
*B R:YOUR ANSWER IS THE SAME AS      |B first time
   THE ONE I GAVE YOU, TRY           |
   AGAIN . . .                       |
*A F: NOW YOU'VE GOT IT. B:15        |A second time
*B R:YOU'RE STILL CONVERTING FROM    |B second time
   CENTIGRADE TO FAHRENHEIT, TRY     |
   AGAIN . . .                       |
*BC F:NOTE THE DIFFERENCE.C:B:OUT    |
*-R                                  |
*-C:                                 |
*                                    |
```

The decision frame contains branching specifications.

The aids to answer processing are phonetic comparison, keyword match and formula equivalence (by algebraic matching).

The calculation mode includes functions, matrices and statistical tables.

2.3.4 TUTOR

TUTOR [Avner and Tenczar, 1969] is the principal author language for the PLATO system developed at the Computer-based Education Research Laboratory of the University of Illinois [Bitzer and Skaperdas, 1970]. The central computer is a Control Data 6400; the system is intended to serve 4000 student terminals on-line at once. The student/author work station is based on a plasma-display panel developed by the PLATO project. Slide images can be superimposed on the text and graphic symbols displayed on the plasma panel. Several special keys, e.g., NEXT, BACK, HELP, and TERM, for lesson control, have been added to a typewriter keyboard.

2.3.4.1 The language

The author language was designed ". . . specifically for use by lesson authors lacking prior experience with computers" [Avner and Tenczar, 1969:1]. The following example illustrates display of text, slides, simple answer analysis, and branching [Avner and Tenczar, 1969:28].

```
UNIT    DAVINCI
NEXT    RUBENS
BACK    INTRO
HELP    DHELP1
WRITE   NAME THE ARTIST WHO
        PAINTED THIS PICTURE -
SLIDE   24
ARROW   1110
ANS     LEONARDO
WHERE   1301
WRITE   THE COMPLETE NAME IS LEONARDO DA VINCI.
SPELL
ANS     LEONARDO DA VINCI
WHERE   1301
WRITE   YOUR ANSWER TELLS ME THAT YOU
        ARE A TRUE RENAISSANCE MAN.
WRONG   WHISTLER
WHERE   1301
WRITE   I HOPE YOU ARE JOKING.
WRONG
WHERE   1301
WRITE   HINT - MONA LISA - HINT
WRONG   MICHELANGELO
NEXT    MREVIEW
```

This is a simple example, mainly intended to show the format of TUTOR statements; the full language has about 70 verbs, called commands, and is functionally very rich, although syntactically it is at the level of assembler language.

The verbs for answer evaluation specify words and characters which may or may not appear in a correct student response. Some spelling correction is performed. The verbs MUST, CANT, and DIDDL are used in the following example [Avner and Tenczar, 1969: command descriptions, DIDDL].

```
UNIT    NURSE
WRITE   DIABETES IS A RESULT OF A MALFUNCTION
        IN THE ...
ARROW   1001
ANS     ABILITY TO METABOLIZE SUGAR
MUST    METABOLISM, UTILIZATION, BURNING, TOLERATION,
        METABOLIZE, UTILIZE, USE, BURN, TOLERATE
WRITE   VERY GOOD
MUST    SUGAR, SUGARS, GLUCOSE, GLYCOGEN
CANT    FAT, FATS, PROTEIN, PROTEINS, VITAMIN,
        VITAMINS, CELLULOSE
WRITE   YOU MUST BE THINKING OF A DIFFERENT DISEASE
DIDDL   ABILITY, CAPABILITY
```

Each lesson has 63 variables to store integers, real numbers, and alphanumeric characters. The variables have fixed names, e.g., I23, A10, and F1. The partitioning of the 63 variables into the three data types is controlled by the author.

2.3.4.2 The operational environment

A student gains access to the system simply by typing his name. He is then resumed where his previous session finished.

An author moves a work station into author mode by pressing the TERM key and entering a password. There are eight author commands, called options [Avner and Tenczar, 1969: Chapter 9]. As TUTOR statements are keyed by an author, they are stored on disk, without any checking by the system. The READIN command initiates a batch compile, at the end of which errors are displayed. To correct errors, an author DELETE's the lesson just read in, issues the command EDIT to make his changes, and, having made them, he re-issues READIN. The system allows only one READIN request at a time; it is thus not a truly multiple-author system.

2.4 Discussion

2.4.1 Coursewriter is the only system which can be said to be production-oriented. If a CAI system is to deal with a large number of real students, with varying motivation, then it must do more than provide rudimentary facilities for student execution and course material preparation. To move from experimental to production status, a system should add facilities for:

(1) registration of students and maintenance of student records;

(2) performance recording;

(3) concurrent use by several authors and several students;

(4) protection of instructional programs from tampering by students and authors;

(5) minimizing the effect of system breakdown on user performance and attitudes.

A fundamental point to recognize is that an instructional program is a non-terminating program. Thus the run-time environment of each student, which may require a large amount of storage for its representation, must be carried over from one session to the next. The implications of this pervade almost all aspects of the operational environment of a production system.

2.4.2 All of the languages have neglected the power of a computer for character string manipulation for two significant tasks - presenting text and specifying answer sets. String manipulation is of course used in answer processing. The simple ability to name a character string would reduce effort when the same text message is used repeatedly. Incorporating string expressions and operations, such as concatenation, in the language further reduces coding effort. Not only is effort reduced but run time efficiency and readability are enhanced.

For example, in a PL/I-like language, if a character string variable named ANSIS has the value 'THE ANSWER IS ', then the display text statement

```
DT ANSIS||'BASE'
```

would print

```
THE ANSWER IS BASE
```

2.4.3 Instructional programs are continually being revised by their authors - during debugging of the first version and as feedback from students is received. Powerful editing and debugging facilities should therefore be a major part of the operational environment. Instead we find minimal editing facilities, and debugging often can only be done by signing on in student mode. If initiating an execution takes more than minimal effort and response time then an author will perhaps tend to spend his time <u>visualizing</u> what his program will do rather than <u>seeing</u> what the student will see.

Most of the systems are typewriter based. The transient image of a CRT terminal allows more natural editing than a typewriter terminal. However, more could be done for the author using a typewriter terminal, e.g., context editors, than is being done. Moreover, the distinction sometimes made between major and minor revisions is artificial.

2.4.4 Inspection of source listings of instructional programs written in these author languages reveals many extraneous symbols and much unnatural syntax. Moreover, the languages do not conveniently illustrate the structure of lessons; FOIL is an exception in this respect. The properties of the underlying computer system or language translator often intrude upon the language. Examples are:

(1) the subroutine linkage mechanism in Coursewriter is exactly the basic machine operation of branch and return on a register;

(2) the @ symbol to denote the counters in WRITEACOURSE stems from a translator deficiency;

(3) labels are defined in FOIL by preceding the label identifier with the symbol :. This device is used to simplify the scanning algorithm of the translator.

(4) parentheses are not allowed in arithmetic expressions in TUTOR. This restriction simplifies the parsing algorithm of the translator.

2.4.5 Because new requirements in author languages are continually being identified, almost all of the language designers acknowledge the need for extensibility in their languages. Those existing systems which try to attain this do it in two ways:

(1) provide a linkage to subroutines written in some other language (assembler in Coursewriter, FORTRAN in FOIL);

(2) implement the language translator in a high-level language so that rewrites are more feasible.

While (1) has the advantage that it does not disturb existing system code, it has the disadvantage that it involves an author in a language with which he is usually unfamiliar. Moreover, the syntax of a subroutine linkage - CALL with a list of parameters - is not usually a natural syntactical specification. The second method has the disadvantage that some of the most intricate system code, namely the translator, has to be changed, and changed in an *ad hoc* manner. Neither method is satisfactory; what appears to be needed is

(1) flexibility which allows extension naturally at the syntactical level, and

(2) a highly systematized implementation of the language translator so that changes can be incorporated according to some formal model.

CHAPTER 3

THE DESIGN PHILOSOPHY UNDERLYING DIAL

This chapter discusses the design philosophy formulated early in the project.

3.1 Motivation

3.1.1 The cost of preparing instructional programs

The cost of preparing instructional programs is high - costs exceeding $100,000 for a one-semester course are not uncommon, for example [Hansen, 1969]. The number of hours of author time to produce one (terminal-time) hour of course material is high. Estimates of 200 or more are reported [Bunderson, 1970]. A major part of this cost is due to extremely time-consuming iterations on the testing and revising phases in course preparation. One wants to reduce these costs. The effects on the total cost of each of the author language and the operational environment for instructional programming have been separated for attack by the CAI System.

3.1.2 Difficulty of use of existing languages

As can be seen from Chapter 2, the syntactic awkwardness of existing languages and the clumsiness of command in existing systems make it difficult for an author to concentrate on his main task, that of preparing instructional material.

3.1.3 Lack of effectiveness of existing languages

To be effective, the semantics of an author language should enable an author to use the unique capabilities of a computer. If these memory, file, and decision capabilities cannot be used, the resulting instructional programs are no different from conventional Programmed Instruction material and the systems deserve the name "expensive page turners" given by Oettinger [1969].

3.1.4 Need for understanding of languages for man-machine systems

Designing a language for CAI and the experience gained from evaluating it and its interactive programming environment should shed some light on the more general problem of man-machine communication. Moreover, the system, once built, could be a valuable research tool for evaluating such languages if the language implementation admits of easy language modification.

3.1.5 Conclusions

These motivating factors lead to two major requirements -

ease of use and modifiability.

3.2 A priori design decisions

3.2.1 The system should be interactive at course preparation time

The benefits of interactive, or conversational, programming are well known. This mode of programming is particularly economical when a program is being changed frequently. This is exactly the situation in CAI, where revision of course material is continually taking place as an author receives feedback from students taking his course.

Moreover, by "seeing what the student sees" as he composes, an author is subject to the same restrictions, e.g., line length, response time, and noise, as his students.

3.2.2 Authors will be experienced teachers

This affects the quality of instructional programs more than the design objectives of DIAL.

3.2.3 Authors will be experienced computer programmers

This is a major departure from the assumptions of most CAI languages. It is justified on two grounds:

(1) pragmatically, it describes the situation of the first subject matter to be taught on the system;

(2) constructing instructional algorithms is just like constructing other algorithms; algorithmic technique cannot be avoided, and it is better and no more costly to learn it for programming in general than for just instructional programming.

But the author inexperienced in computer programming is by no means excluded or ignored. Consider the following.

(1) DIAL is meant to be a language simpler than a language of the complexity of FORTRAN. Its level of simplicity should be comparable to BASIC.

(2) An author, whether experienced in computer programming or not, has to learn the CAI-oriented features of a language which is new to him.

(3) In trying to cater to the inexperienced, a language designer is strongly tempted to over-assist. However, an author preparing any non-trivial instructional program in any new author language, will soon get beyond the stage where assistance with language

mechanisms is needed. An author's facility in using the language should not be underestimated.

(4) There are well-known techniques for assisting learner programmers, e.g., the default concept.

3.2.4 Computer programming will be favored if a subject-matter-dependent design decision arises.

3.2.5 The language should favor the tutorial mode of author-controlled CAI rather than drill-and-practice.

3.2.6 The slide screen, rather than the CRT, will be the main vehicle for the presentation of fixed textual information.

Since color and graphic symbols can be used, the information storage capacity of a slide is high. Note that this assumption obviates the need for picture-drawing facilities in the language.

3.2.7 The basic system approach is CAPI, not question-answering.

In an attempt to improve on the "intelligence" of existing systems as they appear to their student users, time was spent, and the temptation was strong to spend much more, on exploring existing experimental question-answering systems. These systems have been built in research efforts in natural language processing and artificial intelligence [Simmons, 1970].

However, existing systems are indeed experimental and moreover have required major programming efforts to implement. So rather than follow this route, the decision was made to build a well-engineered system, of less ambitious goals. With the generality of a programming language, the tools are provided for an author to produce intelligent instructional programs.

3.2.8 There will be no coder for an author.

I have observed in Coursewriter installations that placing a coder (sometimes called an instructional programmer) between an author and his instructional program generally results in programs that disappoint the author. Instructional programs are, by necessity, representations of algorithmic processes. Too often in this environment, authors describe a concept to a coder and then vaguely define the instructional logic framework in which it is to be presented, without appreciating the algorithmic nature of the problem. For this reason I believe that an author must face, at first hand, the task of structuring his microscopic concepts.

3.2.9 The work station will be CRT-based and fast.

Terminal speed will be sufficient to fill the CRT screen in less than 5 secs (an 800 character CC-30 screen served by a medium-speed communications line -- 2400 bits per second -- is filled in 2 1/3 seconds). An author language designed for a low-speed line (a teletype-speed line fills a CC-30 screen in 80 seconds) would have a different flavor.

The terminal will be CRT-based, so if hard copy is needed, an auxiliary mechanism for providing it will have to be devised as part of the system.

3.3 Objectives for the language per se

3.3.1 First and foremost it should be a programming language.

DIAL must be a language for describing algorithms. Thus it should

allow symbolic names for entities manipulated,
provide a subroutine facility,
have the statement types expected in algorithmic languages,
allow user-defined functions, and
provide a library subroutine facility.

Although CAI workers differ greatly in their approach to using the properties of a computer, there is an agreed-upon requirement for providing individualized instruction. Thus there is the need for author tools for answer analysis and decision making (at the frame level and globally across a course). The need clearly emerges for the author language to have the generality of a programming language.

3.3.2 Special CAI-oriented operators

Because a theory of instructional program writing has not yet been developed, it is not possible to design, or even recognize, an optimal author language. There is, however, a generally accepted set of CAI-oriented commands and utilities. This set should be a part of any author language.

Obviously, given the current level of understanding of the process of writing instructional programs, this set must be embedded in a very flexible system, which can adapt to a variety of different writing techniques.

3.3.3 A general CC-30 display users language

It is anticipated that there will be research (unrelated to CAI) at the University where there will be a need to write programs which use the CC-30. It would thus be useful if DIAL could serve as a general display users language.

3.3.4 Portability

The system should be able to run on computer installations other than the University's.

3.3.5 Advantages of a high level language

Sammet [1969] lists the following six advantages:

ease of learning;
ease of coding and understanding;
ease of debugging;
ease of maintaining and documenting;
ease of conversion;
reduced elapsed time for problem-solving.

My final design objective, which may appear to be obvious, is that these advantages of a higher level language will in fact exist in the final product.

3.4 Objectives for the interactive programming system

The operational environment has as much bearing on ease of use for an author as the language itself. The chief components are the command language and its implementation.

The design objectives for these are as follows.

1. An author should not be aware of the translation from his DIAL statements to machine language. He should feel as if he is programming a "DIAL machine", that is, a machine which directly executes his statements without the need for translation.
2. The CAI System, while in author mode, should at every step try to anticipate an author's next move and position the CRT cursor accordingly.
3. The language processor should be an incremental compiler, not just a fast batch compiler entered interactively. The system could then maintain a consistent response time even when changes to existing source are made.
4. When the known properties of the language, environment, and application dictate system actions that are normally user-specified in a general-purpose system, the CAI System should handle them automatically.
5. Diagnostic messages given by the system should specifically identify the location and type of errors, not just signal that an error has occurred.
6. The CAI System should be responsive to the experience-dependency of an author. For example, the explanatory level of diagnostics given him should decrease as he becomes more familiar with the mechanisms of using

the system.

3.5 Why not adapt a general purpose programming language?

The notion of adapting a well-proven general purpose programming language, such as PL/I, APL, FORTRAN, BASIC, or CPS, was not rejected casually.

For this discussion, the term base language describes the general purpose language, and the term CAI language describes the base language augmented by a set of CAI-oriented routines.

The advantages of adapting a base language include the following.

1. I would not need to write a compiler. Obviously, the routines making up the augmentation must be written, but the programming involved is generally easier than compiler writing.
2. The users of the system would have a well-proven implementation without compiler maintenance responsibilities.
3. A large body of subroutines written in the base language would be available.
4. Authors who know the base language may take less time to learn the CAI language.
5. One would have the generality argued for in section 3.3.1.

In my opinion, the advantages are outweighed by the following disadvantages.

1. Deficiencies in the base language

This impacts writing both augmentation routines and instructional programs in the CAI language. Input/output facilities are not oriented to terminals, particularly CRT's. The absence of file input/output in APL is a severe deficiency. APL and FORTRAN do not have good string handling facilities.

2. Deficiencies in the base language implementation

If the base language is not interactive then one is faced with conversion. The systems programming effort required to convert a base language batch compiler to an interactive incremental compiler would probably be as much as that required to build a compiler for a new author language.

Because the ratio of execution to modification is high, one needs compilative execution to give fast response and low cost for student mode. However, most of the interactive implementations are interpreters.

3. The operational environment is compromised

If there is an interactive implementation of the base language available, e.g., APL/360 or CPS-PL/I, then there is an opportunity to use its command facilities for the

CAI application. But then instructional programs are just like any other programs in the system - a student must load, initiate and terminate a program just as an author, or any other user, does. Other activities, such as student record file processing, performance recording, dumping against system failure, and proctor actions, must also be handled within the existing framework of the host environment. My design philosophy requires an integrated system serving students, authors, and proctors. This cannot be achieved with the general purpose interactive systems available to the project. A special purpose subsystem must be built or the desired operational environment compromised.

4. Overhead

The base language would contain many language features which would never be used by an author. The overhead which results would be felt mainly at compile time, as extra memory space and response time when compiling in author mode. This effect is relatively small.

5. Difficulty of use

Invoking the special CAI facilities, in most languages, would be done by a CALL statement.

This not only adds superfluous code in an instructional program, which can cause readability to deteriorate, but, more importantly, the syntax of the CAI language is awkward to the CAI user. For example,

```
CALL RESUME;
CALL FRAME;
CALL SHOWAS (Note ||'The variable has been used
    before.',A,B);
    .
    .
    .
CALL MATCH (PAT('¢DO ¢=¢'),PAT(X||Y), X||Y,L);
```

Not all base languages, however, require invocation by CALL. APL has the cleaner function invocation, but parameter passing is awkward. PL/I also has user-defined functions, but a function is invoked by the appearance of its name in an expression. This will, for some operations, result in unnatural syntax at the invocation point. Consider a user-defined function for reading. One would prefer to say:

```
READ;
```

but the function name must be in an expression, so one is obliged to say:

```
ANSWER = READ;
```

For control structures, e.g., REPEAT-UNTIL and UNREC in DIAL, even more additional code is needed in the form of labels and GO TO statements.

The macro pre-processor of PL/I (the "compile-time facilities") does provide a solution - a pre-processor pass could substitute correct PL/I syntax for user-oriented syntax. However, when the design decision was made there were no production-status interactive systems providing the macro facility.[1] Moreover, user errors would not be detected until the PL/I translation stage.

My reasons for not adapting a general purpose language can be presented another way. Those requirements for a base language and its implementation which would make adaptation my preferred approach are:

(1) the base language to be PL/I;

(2) the availability of a good incremental compiler;

(3) a well-designed command language for the incremental compiler;

(4) the availability of the PL/I macro facility for implementing the CAI specialized operations with most natural syntax;

---

[1] Since that time, TSO, the Time Sharing Option of Operating System 360 with the MVT configuration, has been announced and is available at TUCC. However, it presents the same problems for the CAI application as do APL and CPS: the operational environment would be compromised.

(5) the operating system to contain a user interface language allowing a user program to execute system commands within a program. As an example of the function, not the syntax, needed, consider an APL program (which is not in fact valid in APL/360):

```
∇ CAI
   .
   .
   .
  STUDENTΔID ← ⎕
  STUDENTΔWS ← SEARCH STUDENTΔID
  )LOAD STUDENTΔWS
   .
   .
   .
  NEXT: EXECUTE
  SAVEΔCOUNT ← SAVEΔCOUNT + 1
  → (SAVEΔCOUNT < 3)/NEXT
  )SAVE STUDENTΔWS
  SAVEΔCOUNT ← 0
  →NEXT
   .
   .
   .
∇
```

With such a facility one would be able to layer the CAI command language on top of the more general, and hence complicated, command language of the base language's operational environment.

CHAPTER 4

THE DIAL LANGUAGE

4.1 Introduction

This chapter describes the author language itself; the command language used by an author while interactively programming in DIAL at a work station is covered in Chapter 5. Hence this chapter specifies the language in which an author writes an instructional program, which is independent of whether he programs interactively or in batch mode from cards were such a facility provided. Another reason for separating the descriptions of DIAL and the command language is that DIAL is the variable part -- changes in the language are implemented by the Translator Writing System described in Chapter 6.

Layout of the chapter

Since this chapter is intended to serve as a guide to using the language, a set of language specifications alone would be inadequate. The development of the chapter is as follows. Section 4.2 presents enough of DIAL to allow complete programs to be written, but with the simplification that all student answers are recognized by exact matching. The remaining sections gradually introduce the full facilities of the language.

The last section, 4.16, DIAL specifications, is a summary of DIAL and can be used for reference purposes by authors experienced in the language.

Since the chapter aims to help a prospective author learn DIAL, rigor is traded for clarity in some sections. The specifications section, however, is rigorous.

A DIAL machine

The design of the language and its operational environment is such that an author can take the view that he is programming a "DIAL machine." This machine directly executes DIAL statements without the need for translation. Thus, as a policy, this chapter avoids referring to the compiler for DIAL.

Color messages

The CRT can display characters in green, red, blue, or yellow. This useful facility enables an author to highlight portions of a CRT message and provide color cues to message content. Because color selection is done in the operational environment, not the language, it is not treated here.

4.2 Writing a simple instructional program

Consider the following segment of an instructional program dealing with logical operations on bit strings:

```
1       SHOW 'If the bit strings B and C contain
          110 and 011 respectively, what is the
          value of A after the execution of A=B&C?'
2 BACK:MATCH '010', OK
3       MATCH '111', NOK
4       SHOW 'Wrong, try again.'
5       GOTO BACK
6 NOK:  SHOW 'No. By definition 0 & 1 is always 0.
        Your answer is correct for A=B|C.
        Try again.'
7       GO TO BACK
8 OK:   SHOW 'Right.'
```

Example 4.1

Statement 1 presents a question by displaying the text

If the bit strings . . . of A=B&C?

on the CRT; statements 2 and 3 specify the student responses anticipated, together with the actions to be taken for those responses. Thus, if the student answers 010 the program branches to the statement labeled OK which displays 'Right.'

<u>Statements</u> are the units of operation within the language. They will normally be executed consecutively as written. However, this sequence of operations may be broken by branching statements. The MATCH and GOTO

statements in Example 4.1 are branching statements. A statement may be optionally prefixed by a label, as statement 6 is.

Identifiers, or names, are created by an author to identify program units in a DIAL program. They have no inherent meaning but serve for the identification of variables, labels and subroutines. An identifier is a sequence of upper or lower case letters and digits, not exceeding ten characters, beginning with a letter, e.g.,

X Cl cardformat BACK clw2

Only identifiers for labels occur in Example 4.1. An identifier naming text, for example, could be R, and so if R had been set previously in the program by the assignment statement

```
R <- 'Right.'
```

then the statement

```
OK: SHOW R
```

would be equivalent to statement 8 in Example 4.1.

DIAL statements are free form. Blanks may be used freely throughout a statement. A blank is needed to separate two tokens in a statement if there is no other delimiter which the DIAL machine can use to determine the separation. For example, X+Y is equivalent to X + Y but GOTOBACK is not equivalent to GOTO BACK. Any number of blanks may appear wherever one blank is allowed.

Composite operators, e.g., <- and ¬= cannot contain blanks.

Unrecognized responses

In the example, only the responses 010 and 111 are recognized. Thus, by the program sequencing rules, any other response will cause statements 4 and 5 to be executed, (with the possibility of a continuous loop: 2, 3, 4, 5).

The *UNREC statement* is a branching statement specifying program action in case unrecognized responses are received. The format of this statement is defined using the notation (Figure 4.1) to be used from now on.

UNREC-statement:

Format:[1]

```
UNREC   label [,label] . . .
```

Action:

The i^{th} unrecognized response to the controlling SHOW-statement will cause a branch to the i^{th} label in the UNREC label list.

---

[1] In this chapter, the first time a statement is presented, usually an abbreviated form will be given. For example, the symbol * can be an item in an UNREC label-list. The chapter progressively develops the full format for each statement.

A uniform system of notation is used to define the format of each DIAL statement. The notation is not a part of DIAL; it is a metalinguistic device to describe the structure of DIAL statements and can be used to describe most programming languages. It indicates the order in which the elements may (or must) appear, the punctuation that is required, and the options that are allowed. The notation is a subset of that used in IBM PL/I publications [IBM 1970b, Section A].

A notation variable names a general class of elements in the language, e.g., label, text-constant, frame-name, and is one of the syntactic units. Other syntactic units are DIAL verbs, e.g., UNREC, punctuation, e.g., a comma, and special characters, e.g., +.

Syntactic units are combined by juxtaposition, braces, and square brackets as follows.

{ } - vertical stacking of syntactic units indicates that a choice is to be made, e.g.,

$$\text{DCL identifier} \left\{\begin{matrix}\text{TEXT}\\ \text{SLIDE}\end{matrix}\right\}$$

[ ] - square brackets denote options. Anything enclosed in brackets may appear one time or may not appear at all. For example,

ENDLESSON [lesson-name]

indicates that a lesson-name is optional in an ENDLESSON statement.

. . . - three dots denote the occurrence of the immediately preceding syntactic unit one or more times in succession. For example,

[,label] . . .

A label preceded by a comma may or may not occur since it is surrounded by brackets. If it does occur, it may be repeated one or more times.

The following example contains each part of the notation.

$$\text{[label:] MATCH text-constant [| text-constant] . . . ,} \left\{\begin{matrix}\text{label}\\ *\end{matrix}\right\}$$

Valid MATCH-statements are

```
d2w: MATCH 'alpha' | 'beta' | 'gamma' , *
     MATCH 'alpha', k3
```

Figure 4.1 -- The metalanguage used to define the syntax of DIAL.

Example:

UNREC L1,L1,HELP,ANS1

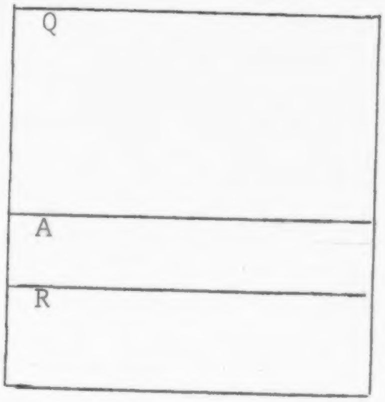
This will cause the statement labeled L1 to be executed after both the first and second unrecognized responses are received. The statements labeled HELP and ANS1 will be executed on receipt of the third and fourth unrecognized responses, respectively. Example 4.2 (statement 4) shows this UNREC-statement embedded in a program segment.

```
1            SHOW     Q
2  BACK:     MATCH    '010',OK
3            MATCH    '111',NOK
4            UNREC    L1,L1,HELP,ANS1
5  L1:       SHOW     'Wrong, try again'
6            GOTO     BACK
7  HELP:     SHOW     'Wrong', LOGIC4,
                      'Now try again'
8            GOTO     BACK
9  ANS1:     SHOW     'No. The answer is 010'
10           GOTO     NEXT
11 NOK:      SHOW     B1
12           GOTO     BACK
13 OK:       SHOW     'Right.'
14 NEXT:
```

Example 4.2

QAR screen division

The CRT screen is divided conceptually into three areas: <u>Q</u>uestion, <u>A</u>nswer, and <u>R</u>esponse:



The Q-area is filled by one or more SHOW's presenting a question. When a MATCH is encountered, the cursor is placed at the beginning of the A-area for the student to enter his answer. The author's feedback response appears in the R-area and the cursor is then placed back in the A-area so inviting the student's next attempt. The student edits his previous answer using the inherent editing properties of the CRT.

For Example 4.1, if the student first entered 111, the screen would appear as

```
If the bit strings B and C contain
110 and 011 respectively, what is the
value of A after the execution of A=B&C?

111

No.  By definition 0 & 1 is always 0.
Your answer is correct for A=B|C.
Try again.
```

Showing slides

As well as displaying text, the SHOW-statement shows slides; for example

```
SHOW nesteddo
```

would, if nesteddo is an identifier naming a slide variable rather than a text variable, show that slide number to which nesteddo is currently set.

The format for the SHOW-statement so far in the development is

$$\text{SHOW} \begin{Bmatrix} \text{text-variable} \\ \text{slide-variable} \\ \text{text-const} \end{Bmatrix} \quad \left[, \begin{Bmatrix} \text{text-variable} \\ \text{slide-variable} \\ \text{text-const} \end{Bmatrix} \right] \ \ldots$$

Example: Line 7 of Example 4.2.

Slide variables can be set by assignment-statements; for example,

```
nesteddo <- 2307
```

would cause SHOW nesteddo to show slide number 7 in carousel 23.

Comments

Comments are enclosed between the markers /* and */ and may be placed anywhere in a DIAL program that a blank is permitted. Any characters may be used in a comment except the pair */, which ends the comment. Comments are completely ignored by the DIAL machine, but their use adds to the readability of an instructional program.

Declarations

Two types of variables have been discussed - the slide variable and the text variable. Notice that an identifier naming a variable has the same formation rules whether the variable is slide or character, but its interpretation in, for example, a SHOW operation will be different. Thus the system must know whether it is to show a slide or text.

This property-designating information comes from associating an attribute with each variable. A variable is given an attribute by one of the two following means.

(1) An author specified declaration

A declare statement is used.

Format:

```
DCL identifier attribute
```

Examples:

```
DCL nesteddo SLIDE
DCL Q TEXT
```

A DCL-statement may appear anywhere in a lesson as long as it appears before the first use of the identifier it names.

Other attributes are introduced in later sections of the chapter.

(2) A default declaration

Unless an author specifies an attribute for a variable, it is assumed by default to be TEXT.

A complete program

Brief consideration for the operational environment is all that is needed now to write a complete lesson. The lesson must be named so that an author can refer to it in the CAI System. This is done by the)LESSON command; since Chapter 5 discusses the command language, it will not be treated here.

The action to be taken by the student at the end of a lesson must be specified. This is done with the ENDLESSON statement.

Format:

```
ENDLESSON  [lesson-name]
```

Action:

The following system message is displayed.

```
END OF LESSON
DO YOU WISH TO GO ON TO THE NEXT LESSON?
TYPE YES OR )OFF
```

The last statement executed in a lesson must be this statement.

A set of lessons constitute a <u>course</u> and a student takes the course lessons in sequence. The simplest complete program, then, is a one-lesson course and Example 4.3 shows such a program.

```
/* An example of a complete program */
DCL logic4 SLIDE
logic4 <- 2301
SHOW 'Message', logic4
ENDLESSON
```

Example 4.3

4.3 Defaulting branching in a program

Many programmed actions are repetitive, e.g., the action of displaying 'Right.' and branching to the next part of the course material to be presented. The DIAL default actions help an author by allowing him to indicate that he will take the default action and so need not explicitly program an action himself.

To take advantage of the default branching it is necessary to give the system some help; an author has to put a little more structure[2] on a lesson by organizing it into <u>frames</u>. A frame is a logically self-contained part of a lesson enclosed by

---

[2] This is more structure than usually found in programs written in a general-purpose programming language, but is the norm for computer-assisted programmed instruction.

frame-name: FRAME

and

END frame-name

where frame-name is an identifier. The DIAL statements in a frame are usually ordered to give the following structure:

```
frame-name: FRAME
  present textual information on slides and CRT
  present question
┌▶accept answer
▲ classify answer
└─respond according to answer classification ─┐
END frame-name                                ▼
```

All defaults actions are requested by the symbol * as follows:

MATCH-statement

<u>Example:</u> MATCH '010',*

<u>Action:</u> If the match is successful then

(1) the system displays 'Right.' in green,

(2) the program branches to the next frame.[3]

---

[3] Note that this is the only part in the language in which the "correct answer" has any significance. Some CAI systems give additional special treatment to the correct answer, e.g., an author can mark a particular text as being the correct answer for later automatic display if the student fails the question. Although the distinction is useful, I believe that the important distinction is not between correct and incorrect responses, but between recognized and unrecognized responses.

UNREC-statement

<u>Example</u>: UNREC *,*,L

<u>Action</u>: for each default label:

(1) the system displays the UNREC message in yellow. This message reads

> Your answer was not recognized. It may be wrong, or it may be right in content but wrong in form, spelling or punctuation. Examine your answer and try again.

(2) the program branches back to the first MATCH-statement (the lead MATCH) of the controlling SHOW-statement.

GOTO-statement

<u>Example</u>: GOTO *

<u>Action</u>: The program branches to the next fram.

Example 4.4 shows the program segment in Example 4.2 recoded using the default sequencing facility.

```
            SHOW Q
BACK:       MATCH '010',*
            MATCH '111',NOK
            UNREC *,*,HELP,ANS1
HELP:       SHOW 'Wrong', LOGIC4
            'Now try again'
            GOTO BACK
ANS1:       SHOW 'No. The answer is 010'
            GOTO *
NOK:        SHOW B1
            GOTO BACK
```

Example 4.4

4.4 Classifying recognized responses

Since, for the branching purposes of a particular question, several responses may be equivalent, it is natural to allow more than one argument in a MATCH-statement. The definition of the MATCH-statement is now

$$\text{MATCH} \left\{\begin{matrix}\text{variable}\\ \text{text-const}\end{matrix}\right\} \left[\,\middle|\left\{\begin{matrix}\text{variable}\\ \text{text-const}\end{matrix}\right\}\right] \ldots, \left\{\begin{matrix}\text{label}\\ *\end{matrix}\right\}$$

If at least one of the specified texts in the list matches the student's answer, then the program branches to label or takes the default action if *. Examples are statements 3 and 6 of Example 4.5.

Thus classification of recognized responses for each question takes the form:

$$\begin{array}{llll}
\text{MATCH} & r_{11} \mid r_{12} \mid & \ldots\ldots\ldots, & \text{label-1} \\
\text{MATCH} & r_{21} \mid & \ldots\ldots\ldots\ldots, & \text{label-2} \\
\vdots & & \vdots & \\
\text{MATCH} & r_{m1} \mid & \ldots\ldots \mid r_{mn}, & \text{label-m}
\end{array}$$

where r_{ij} is the j^{th} recognized response in the i^{th} equivalence class. The vertical bar separator between arguments is to be read as or.

```
          Q <- 'Write an expression which means
                "multiply A by B and assign the
                result to C"'
  L1:     SHOW Q
          MATCH 'C = A*B' | 'C = B*A', OK
          UNREC *,L2,REMED /*On the third, go see what */
                           /*help he needs             */
  L2:     SHOW 'No. Type the two variables being
                     multiplied'
          MATCH 'A B' | 'A,B' | 'B A' | 'B,A',NX
          GOTO REMED  /*Unrecognized, so go          */
                      /*see what help he needs       */
  NX:     SHOW 'Yes.  Multiplying them would
               be achieved by A*B.  Try the
               question again '
          GOTO  L1    /*Back to original question    */
  OK:     /* Present alternative correct answers:    */
          SHOW 'Yes.  C = B*A and C = A*B
               are both right.'
          .
          .
          .
  REMED:
```

Example 4.5

4.5 Expressions and assignment statements

We have already encountered the assignment statement- a value is assigned to the variable on the left hand side. However, in the examples so far, the right hand side contained only one item. It is possible to construct an expression on the right hand side containing several items (variables and constants) with operational symbols (operators) linking the items. For example

Y <- A + B

is an assignment statement which upon execution will cause the expression A+B to be evaluated and its value assigned to Y.

The expression A+B is an arithmetic expression. Other expressions in DIAL are text, slide, logical, and comparison expressions. The items combined by operators are called operands. Operators can only act on operands of the same class, i.e., arithmetic operators can only act on operands with the attribute INTEGER.

Examples:

(1) a text expression using the concatenation operator:

a||'Dog'||b

(2) An arithmetic expression:

a*(b + c)

(3) A slide expression to add one to the slide variable THM:

THM + 1

The two classes of expressions called logical and comparison result in truth values and are discussed in later sections.

A complete definition of DIAL expressions is given in the specifications.

Expressions are not restricted to assignment statements; a new rule is now introduced: whenever text can appear in a statement, a text expression may appear. So we have

```
MATCH A||B, L5
```

and

```
SHOW  a||'Dog'||b , THM
```

The general format of the assignment statement is

$$\text{variable} \texttt{<-} \left\{ \begin{array}{l} \text{text-expr} \\ \text{slide-expr} \\ \text{arith-exp} \end{array} \right\}$$

Examples:

```
L5: REPLY <-   ANSIS||'Dog'
    SCORE <-   A * B + 7
    I <- I + 1
```

4.6 The IF-statement for branching

The MATCH-statement is a branching statement which applies tests to the ANSWER register. With the IF-statement, much more general tests can be applied. For example

```
IF QCOUNT > 3 THEN SHOW R1
SHOW R2
```

compares QCOUNT with 3. If the former is greater, i.e., the test is successful, then the statement SHOW R1 is executed and then SHOW R2. If the test is unsuccessful, SHOW R1 is skipped.

Tests to be performed are specified by a comparison-expression. QCOUNT > 3 is such an expression. Its evaluation results in a truth value (1 = true, 0 = false). The format of the IF-statement so far is

```
IF comparison-expression THEN statement
```

It is sometimes convenient to specify a group of statements to be executed if a test is true. For DIAL, statements can be grouped into a DO-group by using the DO and END, e.g.,

```
DO
 X<-X+1
 SHOW R2
 SHOW R3
END
```

The format of the IF-statement is now:

$$\text{IF comparison-expr THEN} \begin{Bmatrix} \text{statement} \\ \text{DO-group} \end{Bmatrix}$$

In a second, more general form, the IF-statement controls the execution of two alternative statements. The simple IF-THEN clause is expanded by the ELSE-clause. If present, it always follows after the THEN-clause. For example,

```
IF A > 2 THEN X <- Y
          ELSE X <- -Y
```

The THEN-clause is executed only if A > 2 is true; the ELSE-clause is executed only if A > 2 is false. As with THEN, a DO-group may follow ELSE.

Since the comparison-expressions have truth values, they can be combined by logical operators which take truth values as operands. Thus

```
(A > 3) | (B < 5)
```

is a <u>logical-expression</u>.

The format now becomes

$$\text{IF} \begin{Bmatrix}\text{comparison-expr}\\ \text{logical-expr}\end{Bmatrix} \text{THEN} \begin{Bmatrix}\text{statement}\\ \text{DO-group}\end{Bmatrix}$$

$$\left[\text{ELSE} \begin{Bmatrix}\text{statement}\\ \text{DO-group}\end{Bmatrix}\right]$$

<u>Typical uses of the IF-statement</u>

(1) For structuring instructional logic

   a. At the frame level

```
IF LENGTH (ANSWER) > 10 THEN GOTO REM4
```

   b. Globally in a lesson

   Let BADCT be a score kept throughout a lesson and PATH2 an integer set to 1 or 0 to indicate whether a particular path was taken. The following statements specify branching based on the variables BADCT and PATH2.

```
IF (BADCT>25) & (PATH2=1) THEN DO
                              SCALE4  <- SCALE4 + 1
                                /*Increment performance   */
                                /*measure.                */
                              GOTO LAB7 /*Remediation     */
                                END
                           ELSE GOTO L5
                                    /*continue in main    */
                                    /*stream.             */
```

(2) For general programming.

a. At the micro level

```
/*Remove the first occurrence of     */
/*the substring AND from the         */
/*ANSWER register : -                */
J <- INDEX (ANSWER, 'AND')
IF J ¬= 0 THEN /*If J=0 then was not in ANSWER  */
ANSWER <- SUBSTR(ANSWER,1,J-1)||SUBSTR(ANSWER,J+3)
```

b. Loop control

```
    /*Loop to show the first 20 slides  */
    /*in carousel 15 : -                */
    DCL SL SLIDE
    SL <- 1500 /*Initialize  */
L1: SL <- SL + 1
    IF SL > 1520 THEN GOTO EXIT
    SHOW SL
    GOTO L1
EXIT:
```

Nesting

IF-statements can be nested, for example:

```
IF A + B THEN
          IF X = Y THEN Z <- 0
                   ELSE Z <- 1

         ELSE
          IF X > Y THEN P <- 0
                   ELSE P <- -4
```

Finally, note the equivalence of the following two statements.

```
MATCH 'ALPHA'|'BETA' , L5
IF (ANSWER='ALPHA')|(ANSWER='BETA')THEN GO TO L5
```

The MATCH-statement is an abbreviated form of an IF-statement with the ANSWER register as an implied comparand.

4.7 Non-exact matching

To confirm the need for some assistance in recognizing responses, consider the problem of answer processing for the question "What is the name of the UNC student newspaper?" The responses

'Daily Tar Heel ', ' Daily Tar Heel', 'DAILY TAR HEEL',
'The Tar Heel', 'The TAR HEEL',
'DAILY TAR Heeel', and 'DAILY Gazette'

can all be viewed as correct, except for the last one. Restricting answer classification to exact matching would obviously place an intolerable burden on an author, no matter how carefully he phrased his questions.

With the simplification (Chapter 3) that sophisticated linguistic analysis of responses will not be provided in the current phase of the CAI Project, the problem of non-exact matching is left mainly to the ingenuity of the author. The approach taken to help the author combines

(a) the provision in DIAL of system-matching-functions (smf's) and (b) requiring the author to observe certain conventions.

The conventions apply to message preprocessing -- a student's typed response always goes through a preprocessor before ANSWER is filled. The preprocessor

(1) strips preceding and following blanks surrounding a typed response, and

(2) squeezes excess blanks between non-blank characters.

This preprocessing <u>always</u> occurs. To deal with the upper case - lower case problem, the DIAL machine's preprocessor has a switch named CASE which, when <u>on</u>, causes all lower case letters in the response to be converted to upper case as ANSWER is being filled.

Thus while CASE is <u>on</u>, matching will not distinguish between upper case and lower case if MATCH-statement items are written in upper case. For example,

```
MATCH  'THE TAR HEEL' | 'DAILY TAR HEEL' , L5
```

would, with CASE switched <u>on</u>, recognize all but the last two in the example.

CASE is set by an assignment statement

```
CASE <-  arith-expr
```

for example,

```
CASE <- 4
CASE <- 0
```

Thus although CASE stores an integer it has only two states, namely, *on* if its value is non-zero; *off* if zero.

A second preprocessor switch, called SQZ, squeezes *all* blanks from a typed response.

System matching functions

The smf's are PAT for pattern matching against the ANSWER register and PEN for using the light pen. The latter is detailed in a later section. The smf's return a truth value and hence are (one-item) logical expressions. To effect branching based on the values returned, smf's can appear in MATCH-statements and IF-statements.

The PAT smf uses the pattern specified as its argument and searches ANSWER for the occurrence of that pattern. A pattern is made up of a sequence of pattern elements separated by a cent symbol, the "don't care" symbol, where any number of noise symbols may appear. For example, PAT('¢*¢+¢') would be true if ANSWER contained ALPHA * BETA + D.

By making answers to questions of the type "Give an expression which multiplies two variables" rather than "Give an expression which multiplies A and B" quite easy to process, the PAT function should encourage an author to go from the specific to the general in his questioning.

Because PAT is quite general it covers simpler functions often found in CAI languages. Keyletter and keyword matching are two examples. Keyletter matching is intended to cope with spelling errors in a student response. For example,

```
MATCH PAT ('¢IDENT¢F¢R¢'), L5
```

would cope with certain misspellings of the word "identifier."

All correct answers, except the last, in the Tar Heel example would be recognized by the keyword matching

```
PAT ('¢TAR HEEL¢')
```

if CASE is on.

Note that the text function INDEX can also be used for keyword matching.

Notice also that PAT has an implied ordering by the order of the pattern elements. So the following two segments are equivalent:

```
(1)   MATCH PAT ('¢TAR¢HEEL¢'), L1
      GOTO L2

(2)   J <-  INDEX(ANSWER,'TAR')
      K <-  INDEX(ANSWER,'HEEL')
      IF J=0 | K=0  THEN GOTO  L2
      IF K > J THEN GOTO L1
      GOTO L2
```

4.8 The sieve

Shortly after Brooks began to use DIAL (Chapter 7) he invented the sieve; it was made possible by the PAT system matching function.

Figure 4.2 gives an example. The actual sieve is contained in statements 224 through 250.

The sieve consists of an ordered set of expected responses. The first is correct; each successor allows one more erroneous element, or conversely requires one fewer correct element. The responses are arranged in order of increasing seriousness of error.

An answer falls through the sieve until it encounters the first response that requires no more correct elements than the answer has.

The feedback for each response is designed to teach about precisely the error that distinguishes that response from the one above it, and it always requires that the student try again.

This correspondence between feedback and sieve level works properly for several reasons. Falling through to that level means the student surely made at least the error addressed. Falling no further means he made a no more serious error. The ordering means that the error addressed is the most serious one he made, however many

```
 200     d2: /*****************************/
             RESUME;
 202       CASEOFF;
 204       S don, CLEAR;
 208                 PINT;
 212 :
 216 :
 220       SAS  CLEAR,
'Write a statement labeled SAM that will

 iterate the group of statemonts it

 controls 50 times.  Use the variable X

 as your index.';
 224 d2m: M 'SAM:DO X=1 TO 50;',d2r;
 228      M 'SAM:DO X=1 TO 50',d2w1;
 230      M PAT('SAM:DO X=1 TO 50¢'),
                                d2w1a;
 232      M PAT('SAM¢DO X=1 TO 50¢'),
                                d2w2;
 236      M PAT('¢DO X=1 TO 50¢'),d2w3;
 240      M PAT('¢DO X=1 TO¢'),d2w4;
 244      M PAT('¢DO X=1¢'),d2w5;
 246      M PAT('¢DO X=¢'),d2w6;
 248      M PAT('¢DO¢'),d2w7;
 250      U *;
 252      SAS
'Your statement should contain DO.',t;
 254      GOTO d2m;
 256 d2w1:S nq,forgotsemi,t;
 257      GOTO d2m;
 258 d2w1a:S extramsg,t;
 259      GOTO d2m;
 260 d2w2:SAS
'A colon should follcw the label.',t;
 262      GOTO d2m;
 264 d2w3:SAS
'The statement should be labeled SAM.',t
;
 266      GOTO d2m;
 268 d2w4:SAS
'The upper limit of the iteration count

 should be 50.',t;
 270      GOTC d2m;
 272 d2w5:SAS
'The word TO separates the starting and

 ending values of the iteration count.',
t;
 274      GOTC  d2m;
 276 d2w6:SAS
'The iteration count should start with 1
',t;
 277      GOTC d2m;
 278 d2w7:SAS
'The iteraticn ccunt expression should

 begin


       X=    ',t;
 280       GOTO d2m;
 282 d2r:  S r;
 400 :
```

1. DIAL VERB ABBREVIATIONS:

M	MATCH
S	SHOW
SAS	SHOWAS
U	UNREC

2. TEXT CONSTANTS NAMED EARLIER IN THE PROGRAM:

extramsg	'You have entered something in addition to or instead of the semicolon which should end a PL/C statement.'	(blue)
forgotsemi	'You forgot the semicolon.'	(blue)
nq	'Not quite. '	(yellow)
r	'Right.'	(green)
t	'Try again. '	(red)

3. don is slide 151, named earlier in the program. It is shown in Figure 5.6.

4. PINT, a temporary addition to DIAL (see 8.3.3.2,2), displays "Press INT to continue when ready." and then waits for INT.

Figure 4.2 -- A DIAL program segment showing a sieve for answer evaluation of the expected response SAM:DO X=1 TO 50;

others there may be. Trying again means the errors are all treated, one at a time. The result is an analyzer whose length and complexity grows linearly with the number of anticipated errors, but which is capable of handling all combinations of them.

Progressive teaching also results from this ordering of the sieve elements. If the student has no idea what the answer should be he will be led, step by step, through the construction of the right answer. Such a trace of the sieve in Figure 4.2 is:

Student Response	Sieve element MATCHed (DIAL statement #)	Feedback
(null)	250	(UNREC message)
(null)	252	Your statement should contain DO. Try again.
DO	248	The iteration count expression should begin X= Try again.
DO X=	246	The iteration count should start with 1. Try again.
DO X=1	. . .	

4.9 The naming statement

The assignment-statement of Example 4.5 assigns a text constant value to the variable Q. This is a useful technique when a particular text is to be used repeatedly, as it saves rewriting the complete text each time. There is a DIAL statement solely for this naming function, which uses = to indicate identity rather than < - which is used to indicate setting of a variable value.

Format:

```
identifier = text-const
```

Examples:

```
obs = 'Observe the slide above.'
pint = 'Press INT to continue.'
```

When an identifier is so used, it is given the TEXT (constant) attribute.

Variables, by definition, require the DIAL machine to keep a separate copy for each student. In contrast, when an author uses a naming-statement he is signalling that the identifier is a name, not a variable-name. Hence only one copy is needed for all students. Substantial savings result in the storage of the DIAL machine.

While such names cannot be used on the left hand side of an assignment statement, they can form text-expressions, e.g.,

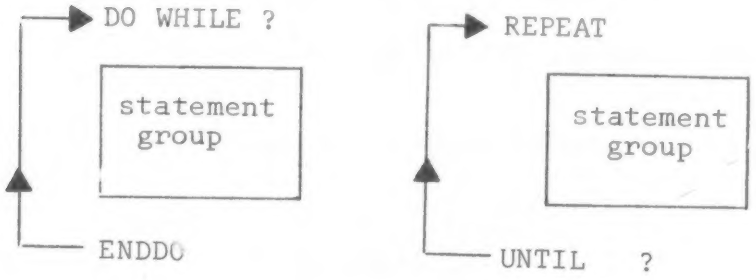
SHOW obs||pint

There is a similar naming facility for slide constants, e.g.,

nesteddo = 2330 .

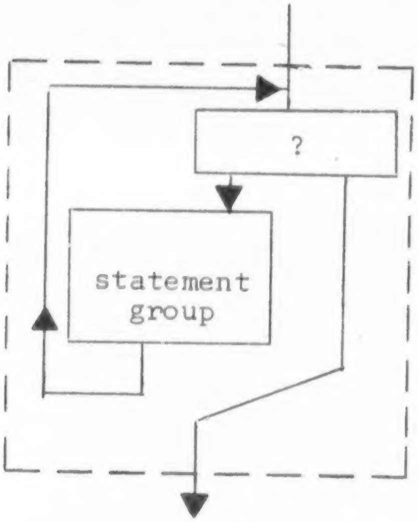
4.10 Repetition constructs

DO-WHILE and REPEAT-UNTIL are available for controlling the repetitive execution of a group of statements. They are motivated in [Dijkstra, 1970]. The constructs


DO WHILE ?
statement group
ENDDO
REPEAT
statement group
UNTIL ?

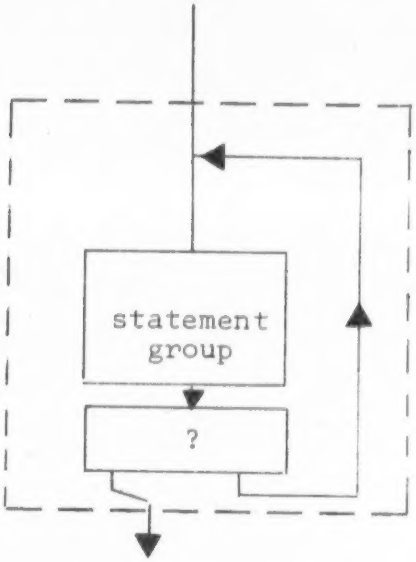
can be diagrammed as follows:

DO WHILE


?
statement group

(leading decision)

REPEAT UNTIL


statement group
?

(trailing decision)

Examples:

(1)

```
/*HARDWARE TEST PROGRAM      */
/*Show all slides in carousel 3:- */
DCL s SLIDE
s <- 300
REPEAT
  S<- S + 1
  SHOW s
UNTIL s = 380
```

(2)

```
pint = 'Press INT to continue'
  .
  .
  .
REPEAT  -
  SHOW pint
UNTIL PAT('')
```

```
(3)     I <-  1
        DO WHILE I <= 100
           SHOW I
           I ← I + 1
        ENDDO
```

Format:

```
        DO WHILE  { comparison-expr }
                  { logic-expr      }

        ENDDO   [do-while-label]

        REPEAT

        UNTIL  { comparison-expr }
               { logical-expr    }
```

4.11 Cathode-ray tube screen formatting

The CRT of the CC-30 can display 800 characters, arranged as 20 rows, each of 40 characters. Two aspects of CRT screen formatting concern the author, (1) the relative positioning of successive screen messages and (2) the relative positioning of characters within a message.

(1) Relative positioning of screen messages

Successive text-expressions in a SHOW-statement are separated by commas. When the statement is executed, each text-expression is evaluated and treated as one screen message. Each screen message is displayed beginning at the next free screen row. For example, if the text-variable GAM contained 'GAMMA' then the statement

```
SHOW 'ALP'||'HA', 'BETA', GAM
```

would result in

```
ALPHA
BETA
GAMMA
```

whereas the statement

```
SHOW 'ALP'||'HA'||'BETA'||GAM
```

would result in

```
ALPHABETAGAMMA
```

The above discussion applies to messages <u>within</u> the Q-area and A-area. The first SHOW-statement of a frame begins at the top of the Q-area. The QAR boundaries are floating and can be changed by assignment to the system variables QVALUE, AVALUE, and RVALUE. For example,

```
AVALUE <- 15
RVALUE <- 17 .
```

These three system variables have default values of 1, 11, and 13. An author must ensure that screen messages fit within the QAR division; overflows will be displayed in the next area, for example, a too large Q-area message will overwrite the A-area.

(2) Relative positioning of characters within a message

The exact format of a screen message is important if the message is a table or list of items, whereas if it is entirely prose, its format is of less concern.

For exact screen formatting, the SHOWAS-statement (to be read "Show as formatted") is used. It has the same syntactic format as SHOW, namely,

$$\text{SHOWAS}\begin{Bmatrix}\text{text expr}\\ \text{slide-expr}\end{Bmatrix}\quad\left[,\begin{Bmatrix}\text{text-expr}\\ \text{slide-expr}\end{Bmatrix}\right]\ldots$$

When a display statement is encountered during execution, the DIAL machine, for each text-expression, evaluates the expression and then

(1) if it is in a SHOWAS-statement the screen message is displayed as it was originally formatted,

(2) if it is in a SHOW-statement the machine formats the message by removing instances of word-breaks over screen lines and then displays the result.

Examples:

Statement executed:

```
112   next:   SHOW  'The slide screen abo
ve is used for presenting the bulk of th
e TEXTUAL MATERIAL.'
```

Result:

```
The slide screen above is used for
presenting the bulk of the TEXTUAL
MATERIAL.
```

Statement executed:

```
250  SHOWAS                                  '
        DIAL has been designed for
          AUTHOR-CONTROLLED CAI'
```

Result:

```
        DIAL has been designed for
          AUTHOR-CONTROLLED CAI
```

Thus characters within text are formatted in two ways. By using SHOW, an author need not be concerned with word breaks as he is typing text. By using SHOWAS he can specify the exact layout of a display.

The following illustrates the display of text constants. Let the text be named by the statement

```
150 studyslide='STUDY THE SLIDE ABOVE AN
D PRESS INT TO CONTINUE.'
```

Then typical uses are the following.

Statement executed:

```
750 SHOW studyslide
```

Result:

```
STUDY THE SLIDE ABOVE AND PRESS INT TO
CONTINUE.
```

Statement executed:

```
850 SHOWAS studyslide
```

Result:

```
                STUDY THE SLIDE ABOVE AN
D PRESS INT TO CONTINUE.
```

Statement executed:

```
950  SHOWAS studyslide,'ALPHA'||'BETA'
```

Result:

```
                STUDY THE SLIDE ABOVE AN
D PRESS INT TO CONTINUE.
                      ALPHABETA
```

4.12 Light pen usage

Instructional programs can be written so that a student indicates his response to a multiple-choice question by pointing with the light pen.

Presenting the multiple-choice question

The multiple-choice items are referred to as light pen targets and must obey the following rules.

(1) Each item must begin with the character *;

(2) Each item must occupy no more than one row;

(3) There can be no more than eight targets.

Figure 4.4 has four targets. An author will normally use a SHOWAS-statement to present his targets in a multiple-choice question.

Recognizing the student's response

When a student points to a target and presses the pen's button, a light pen hit is said to have occurred (the system recognizes only one hit at a time). An author specifies a hit and its branching action by using the system-matching-function PEN in a MATCH-statement or IF-statement. Examples are

```
MATCH PEN(3)|PEN(5),L5
```

and

```
IF PEN(2) THEN SHOW DIAG
```

A complete example of light pen usage is given by the program segment in Example 4.6 and its execution in Figures 4.3 and 4.4.

```
110 SHOW struc4,ref,'ADDRESS on line 5
is followed by STREET, CITY, and STATE,
which are all declared with a greater le
vel number than ADDRESS.  You can see
that STREET, CITY, and STATE are contain
ed in ADDRESS.'
120 SHOWAS  '
     Since ADDRESS has data items in it,
it is:-

     * a major structure
     * a minor structure
     * an elementary name
     *  (none of these)'
130 L1: MATCH PEN(2),*
        MATCH PEN(1),MAJOR
        MATCH PEN(3)|PEN(4),BAD
140 BAD:
     .
     .
     .
180 MAJOR:
     .
     .
     .
```

Example 4.6

```
 1   DECLARE
 2    1   PERSONNEL,
 3     2  NAME         CHAR(24),
 4     2  PHONE        CHAR(9),
 5     2  ADDRESS,
 6         3 STREET    CHAR(25),
 7         3 CITY      CHAR(15),
 8         3 STATE     CHAR(2),
 9    1  YEAR_TO_DATE,
10     2= GROSS        FIXED DEC(8,2),
11     2  TAX          FIXED DEC(7,2);
```

Figure 4.3 -- The slide struc4 used in Example 4.6.

```
Refer to the slide.
ADDRESS on line 5 is followed by STREET,
CITY, AND STATE, which are all declared
with a greater level number than
ADDRESS.  You can see that STREET, CITY,
and STATE are contained in ADDRESS.

     Since ADDRESS has data items in it,
it is:-

     * a major structure
     * a minor structure
     * an elementary name
     *  (none of these)
```

Figure 4.4 -- An Execution of Example 4.6.

4.13 The RESUME-statement

The point at which a student terminates a particular session may not be the best point for him to begin his next session. It may be better pedagogically if he is restarted at a point just prior to the end of the last session. An author therefore places RESUME-statements at suitable points throughout the lesson.

Format:

RESUME

Action:

Causes the CAI System to copy the student's complete status to the RESUMDUMP file.

4.14 Subroutines

4.14.1 Since subroutines in DIAL have the same definition and invocation conventions as subroutines in a general purpose programming language, they will not be described in detail. The specifications section of this chapter defines the CALL, PROC and END statements needed.

There are two types of subroutine procedures - DIAL subroutines and PL/I subroutines.

(1) DIAL subroutines

Example 4.7 is a subroutine, DELETE, to delete all occurrences of specified characters from ANSWER.[4] A sample invocation is

CALL DELETE ('AEIOU')

```
          DELETE: PROC(DELS)
            /*This subroutine deletes from ANSWER     */
            /*all the occurrences of the characters in*/
            /*DELS.                                   */
            /*(Note the length restriction of 10 on   */
            /*DELS imposed by the dimension of        */
            /*vector C)                               */
          DCL DELS TEXT    /*Parameter*/
          DCL L INTEGER  /*Length of, i.e., number of */
                         /*elements in, DELS          */
          DCL C(10) TEXT /*Holds the elements after   */
                         /*unpacking from DELS        */
          L <- LENGTH(DELS)
          I <- 0
OUTER:    I <- I + 1
          IF I > L THEN GOTO RETURN
          C(I) <- SUBSTR(DELS,I,1) /*unpack next element*/
INNER:
          J <- INDEX (ANSWER,C(I))
          IF J=0 THEN GOTO OUTER
          ANSWER <- SUBSTR(ANSWER,1,J-1)||SUBSTR(ANSWER,
                                          J+1)
          GOTO INNER
RETURN:
          END DELETE
```

Example 4.7

---

[4] This subroutine also shows the use of vectors in DIAL.

Example 4.8 is a subroutine, MULTI, to display multiple-choice items for light pen selection and to return the student's choice. A sample invocation is

```
CALL MULTI ('Point to the name of
   the animal that barks:',
  'dog', 'cat', 'rat', a)
```

(2) PL/I subroutines

Subroutines written in PL/I can be called from a DIAL program. This provides a means of augmenting the power of DIAL. It is anticipated that experienced and resourceful authors will use this facility in their answer analysis. Experience will show which PL/I facilities are the most popular; these will then be considered for implementation in DIAL itself.

Invocation of PL/I subroutines is exactly the same as for DIAL subroutines. The subroutine names, however, must be declared with the attribute PLISUB, e.g.,

```
DCL sub PLISUB
```

The subroutines themselves are defined outside of the CAI System.

4.14.2 The name scope rule is as follows. The scope of a declaration is a lesson unless the declaration is within a procedure. Then its scope is that procedure including all contained procedures except those containing another explicit declaration of the same identifier.

```
MULTI: PROC(PREAMB,ITEM1,ITEM2,ITEM3,PENR)
   /*A subroutine to present three multiple-          */
   /*choice items for light pen selection.  A         */
   /*preamble in PREAMB introduces the items.         */
   /*The student response is returned in PENR.        */
DCL PENR INTEGER

/***Present multiple-choice:- **/
   SHOW PREAMB
   ITEM1 <- '* '||ITEM1   /*Prefix *  */
   ITEM2 <- '* '||ITEM2
   ITEM3 <- '* '||ITEM3
   SHOWAS ITEM1,ITEM2,ITEM3

/***Read student response: -  **/
   IF PEN(1) THEN PENR <- 1
   IF PEN(2) THEN PENR <- 2
   IF PEN(3) THEN PENR <- 3
END MULTI
```

Example 4.8

4.15 Input-output synchronization

Output from a DIAL program, effected by the SHOW or SHOWAS statements, consists of a CRT display or the projection of a slide. Input to the program, effected by the MATCH statement, is a typed or penned response.

Output

Although a SHOW or SHOWAS statement may contain several separate text-expressions and a slide, each separated by a comma in the operand list, these separate items are displayed without any time delay between them.[5] There is, however, a time delay equal to the setting of the PAUSE register between two successive SHOW's without an intervening MATCH.

Input

There is no explicit read statement in DIAL; reading is implicitly requested by MATCH-statements. The first MATCH-statement encountered after a SHOW causes the DIAL machine to issue a read - the blue Proceed light (Keyboard Enabled light) comes on and the read is completed when the student sends his response by depressing INT. If the next statement to be executed is also a MATCH statement, it will not issue a read but will perform the answer matching using the contents of the ANSWER register filled by the preceeding MATCH-statement. These rules apply not only to MATCH but also to PEN and PAT.

---

[5] If there is more than one slide, only the one appearing last will remain projected.

To further clarify this synchronization consider the DIAL machine's internal mechanism to effect it. An internal switch named READISSUED is tested by each statement that accesses the student's response. If READISSUED is *off*, then a read is issued and the switch turned *on*, if it is *on* then no read is issued. It is also turned off by a SHOW-statement and an UNREC-statement.

As an example, trace an execution of the following DIAL program segment of nine statements

```
1          SHOW
2          SHOW
3          SHOW
4          MATCH
5          MATCH               ,  L5
6          MATCH
7          UNREC *,*,L3
8  L5:     SHOW
9          SHOW
```

The trace could be

Statement Number	Action
1	show
2	pause
	show
3	pause
	show
4	no pause
	read
	unsuccessful match
5	no read
	unsuccessful match
6	no read
	unsuccessful match
7	show UNREC-message
4	read
	unsuccessful match
5	no read
	successful match

8	no pause show
9	pause show

Note that the sequence which controls the synchronization is the order of execution, not the (sequential) statement numbering. A trace of the following interaction (involving a subroutine call) further exemplifies this point.

```
  1    SHOW
  2    SHOW
  3    CALL PINT
  4    SHOW
  5    SHOW
  .
  .
  .
100    PINT: PROC
       /*This procedure returns control to the          */
       /*calling point when a plain interrupt has       */
       /*been received.  The PAUSE register             */
       /*is zeroed for the duration of PINT so          */
       /*that the proceed request is displayed          */
       /*immediately                                    */
101    DCL SAVEL INTEGER
102    SAVEL <- PAUSE
103    PAUSE <- 0
104 L1:SHOW 'Press INT to continue'
105    M '', L2
106    GOTO L1   /*not plain interrupt */
107 L2:PAUSE <- SAVEL
108    END PINT
```

Trace:

Statement Number	Action
1	show
2	pause
	show
3	
100	
101	
102	
103	
104	pause (of zero seconds)
	show
105	read
	unsuccessful match
106	
104	no pause
	show
105	read
	successful match
107	
108	
4	no pause
	show
5	pause
	show

4.16 DIAL specifications

Since this section can be used for reference purposes, the following listing of section headings may be useful.

4.16.1 Message preprocessing

4.16.2 Statements

(1) Declarations

(2) Input/output to the student

(3) Branching

(4) Assignment

(5) FRAME-statement
(6) DO-group head
(7) PROC-statement
(8) END-statement
(9) CALL-statement
(10) ENDLESSON-statement
(11) RESUME-statement
(12) Naming-statement
(13) Repetition statements

4.16.3 Text manipulation
4.16.4 Expressions
4.16.5 System matching functions
4.16.6 Light pen usage
4.16.7 Default actions
4.16.8 Abbreviations
4.16.9 Restrictions

The metalanguage used to define the syntax of DIAL is a subset of the syntax notation used in IBM PL/I publications [IBM 1970b: Section A] and is given in Figure 4.1.

The CAI System has been designed so that an author can take the view that he is programming a "DIAL machine." This machine, which is diagrammed in Figure 4.5, has the following characteristics:

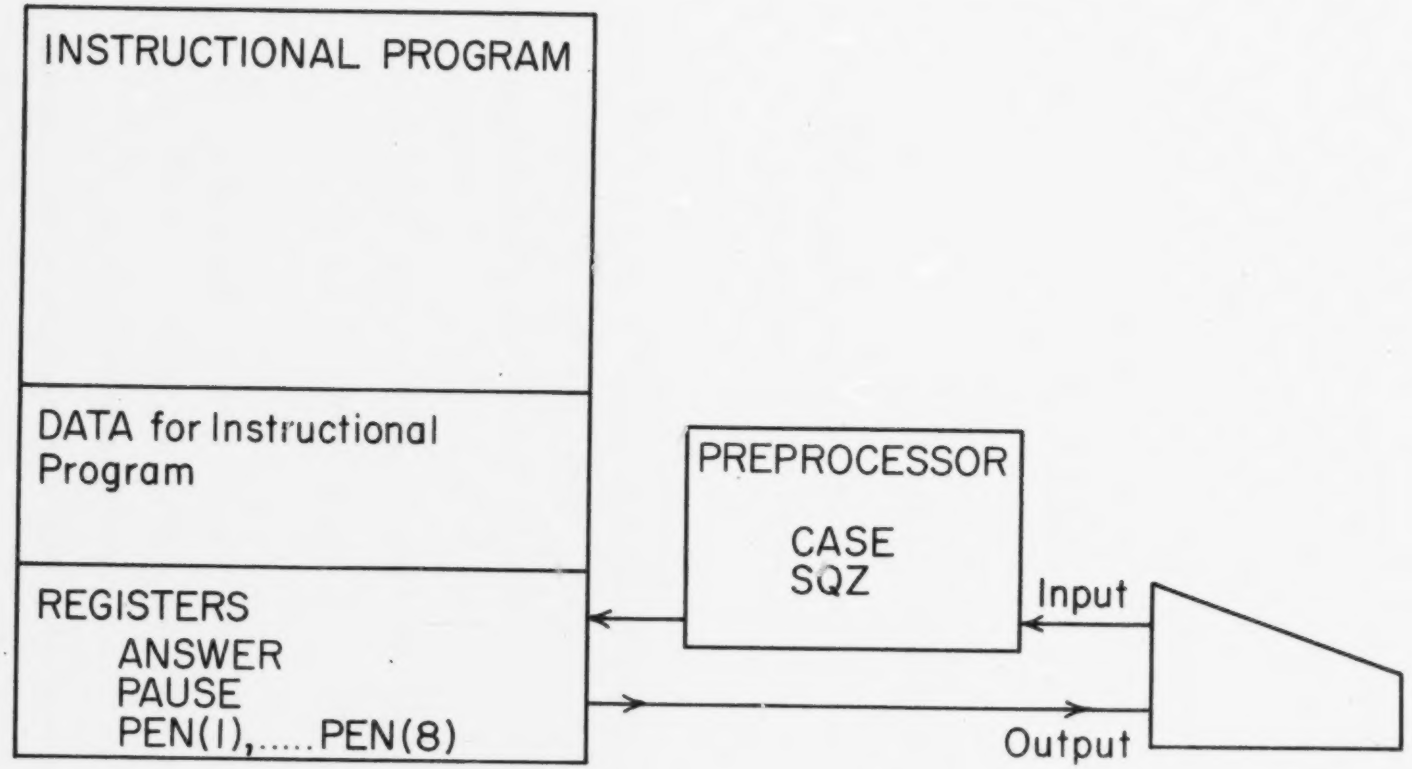

INSTRUCTIONAL PROGRAM
DATA for Instructional Program
REGISTERS
ANSWER
PAUSE
PEN(1),.....PEN(8)
PREPROCESSOR
CASE
SQZ
Input
Output

Figure 4.5 -- A DIAL machine.

(1) it directly executes DIAL statements without the need for translation;

(2) it has a one-level store, which can hold arbitrarily large programs.

This view is possible because the CAI System has been implemented in accordance with the "onion structure" given in Figure 4.6.

4.16.1 Message Preprocessing

A student's typed response is always preprocessed before author-specified answer classification, or matching, begins.

Automatic preprocessing

The preprocessor always does the following:

(1) strips preceding and following blanks surrounding an answer

(2) squeezes excess blanks between non-blank characters in an answer.

Example:

Let b̸ represent the blank character. If the answer typed by a student is

b̸Xb̸=b̸b̸Ab̸b̸+b̸Bb̸;b̸b̸

then the ANSWER register contents after automatic preprocessing will be

Xb̸=b̸Ab̸+b̸Bb̸;

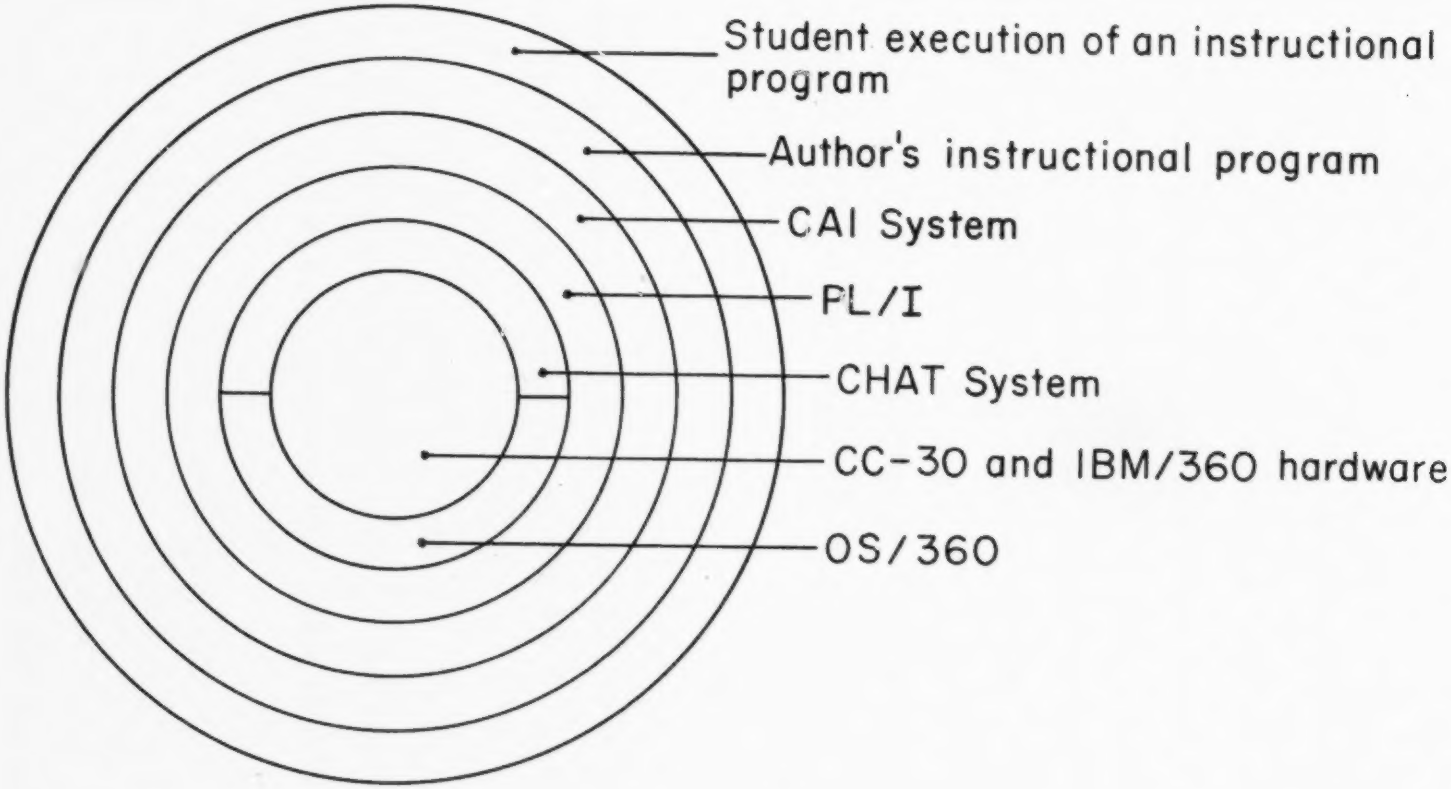

Student execution of an instructional program
Author's instructional program
CAI System
PL/I
CHAT System
CC-30 and IBM/360 hardware
OS/360

Figure 4.6 -- The "onion structure" implementation of the CAI System.

Switchable preprocessing

When the CASE switch is on, all lower case letters in a typed response are converted to upper case as ANSWER is being filled. CASE is in the on state by default.

When the SQZ switch is on, all blanks are squeezed (removed) as ANSWER is being filled. SQZ is in the on state by default.

The SQZ and CASE switches are actually registers holding INTEGER data in the DIAL machine. The on/off states are

on	register non-zero
off	register zero

Since both SQZ and CASE are system variables, their settings can be changed by assignment statements, e.g.,

```
CASE <- 1
SQZ  <- 0
```

Although these registers are set by assigning integers to them, when they are read they return a truth value. Thus

```
IF CASE THEN CASE <- 0
```

and

```
IF ¬ SQZ THEN SHOW MSG
```

are valid, but the following is not:

```
IF CASE = 10 THEN CASE <- 0
```

4.16.2 Statements

A DIAL lesson, or program, is constructed from basic program elements called statements. Statements make up larger program elements: DO-groups, frames, procedure-definitions, REPEAT-UNTIL blocks, and DO-WHILE blocks. A DO-group is a sequence of statements headed by a DO-statement and terminated by a corresponding END-statement. A frame is delimited by a FRAME-statement and an END-statement. A procedure-definition is delimited by a PROC-statement and an END-statement.

Execution passes sequentially into and out of a frame, whereas a procedure must be invoked by a CALL-statement.

Comments are enclosed between the markers /* and */ and may be placed anywhere in a DIAL program that a blank is permitted. Any characters may be used in a comment except the pair */, which ends the comment. Comments are completely ignored by the DIAL machine.

Definitions of each statement follow.

(1) Declarations - the DCL-statement and attributes

Syntax:

$$\left\{\begin{matrix}\text{DCL}\\ \text{NEW}\end{matrix}\right\}\text{identifier attribute-specification}$$

Action:

The statement is used to declare explicitly the attributes of identifiers. The attributes in DIAL are:

TEXT
SLIDE
INTEGER
PLISUB
GLOBAL
LABEL

The attribute specification contains the attribute name and, when allowed, a vector dimension.
A DCL-statement may appear anywhere in a lesson as long as it appears before the first use of the identifier it names.

(a) Vector identifiers

An identifier may name a vector (a one-dimensional array) of variables of the same attribute. The dimension of the vector precedes the attribute-name and is enclosed in parentheses. For example,

```
DCL SCORES(10) INTEGER
```

(b) Label vectors

These vectors allow a "computed-GO-TO" facility in DIAL and can be used only in this way. There are no label variables as in PL/I.

(c) Constants and slide constants

When an identifier appears in a naming-statement it is given the TC or SC attribute; TC and SC cannot appear in a DCL-statement.

(d) GLOBAL

Identifiers with this attribute have the implied attribute INTEGER and have name scope across all lessons in a course.

Note that the)INCLUDE facility (Chapter 5) provides a means of giving text <u>constants</u> global scope.

(e) PLISUB

If arguments are to be sent to the PL/I subroutine then the attributes of their corresponding parameters must be given. If a parameter is a vector then this is indicated by appending (#) to the attribute. For example,

```
DCL P PLISUB (TEXT,INTEGER(#),TEXT)
```

would define a subroutine with three parameters. A sample invocation is

```
CALL P (ANSWER,SCORES,REPLY)
```

(f) Summary

$$\left\{\begin{array}{l}\text{DCL}\\\text{NEW}\end{array}\right\}\ \text{identifier [(dimension)]}\ \left\{\begin{array}{l}\text{TEXT}\\\text{SLIDE}\\\text{INTEGER}\\\text{GLOBAL}\end{array}\right\}$$

$$\left\{\begin{array}{l}\text{DCL}\\\text{NEW}\end{array}\right\}\ \text{identifier (dimension) LABEL}$$

$$\left\{\begin{array}{l}\text{DCL}\\\text{NEW}\end{array}\right\}\ \begin{array}{l}\text{identifier PLISUB}\\\text{[(parameter-attribute-list)]}\end{array}$$

(2) Input/output to the student

a. SHOW-statement

<u>Syntax</u>:

$$\text{SHOW}\ \left\{\begin{array}{l}\text{text-expr}\\\text{slide-expr}\\\text{arith-expr}\end{array}\right\}\ \left[,\left\{\begin{array}{l}\text{text-expr}\\\text{slide-expr}\\\text{arith-expr}\end{array}\right\}\right]\ \ldots$$

Action:

Each expression is evaluated and then sent as an output to the terminal.

(a) text expressions

each character string is treated as a separate screen message and is displayed at the beginning of the next free CRT row. Characters appear with no words broken over rows.

(b) slide expressions

the slide is shown, and remains projected until the next slide action.

(c) arithmetic expressions

the result of the evaluation is converted to character form and displayed on the next free row.

Examples:

```
SHOW 'Central Processing Unit'
SHOW ANSIS||'UNIT', 'Refer to the
        slide above', DIAG2
```

b. SHOWAS-statement

(to be read as "Show as formatted")

Syntax:

$$\text{SHOWAS} \left\{\begin{matrix}\text{text-expr}\\ \text{slide-expr}\\ \text{arith-expr}\end{matrix}\right\} \left[\, , \left\{\begin{matrix}\text{text-expr}\\ \text{slide-expr}\\ \text{arith-expr}\end{matrix}\right\}\right] \ldots$$

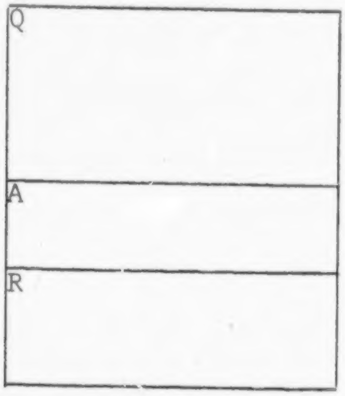
Action:

As for SHOW-statement except that no word-break removal is performed on text. Thus characters are formatted exactly as they appeared when originally entered.

c. CRT screen formatting

QAR screen division

The CRT screen is divided conceptually into three areas: Question, Answer, and Response:



The Q area is filled by one or more SHOW's presenting a question. When a MATCH is encountered, the cursor is placed at the beginning of the A-area for the student to enter his answer. The author's feedback response appears in the R-area and the cursor is then placed back in the A-area so inviting the student's next attempt.

The QAR boundaries are floating and can be changed by assignment to the system variables QVALUE, AVALUE, and RVALUE. Their default settings are 1, 11, and 13.

Relative positioning of successive screen messages

Each successive text-expression results in a SHOW-statement or SHOWAS-statement is displayed beginning at the next free row in the Q-area or R-area.

Relative positioning of characters within a message

This positioning is done by the system according to whether SHOW or SHOWAS is used.

d. Duration of slide showing

A slide shown will remain projected until one of the following occurs:

(1) the system slide constant RMV is shown

(2) a new frame begins execution

(3) another slide is shown

(4))OFF is received.

Note that RMV is an opaque slide occupying position 0 of each carousel. Thus only positions 1 through 80 are for author use.

e. Reading a student's response

The keyboard will be enabled, so inviting a student's typed or penned response, whenever the first MATCH-statement after the controlling SHOW-statement, or SHOWAS-statement, is executed. The response is then read and processed by MATCH-statement(s).

(3) Branching

a. MATCH-statement

Syntax:

$$\text{MATCH} \begin{Bmatrix}\text{text-expr}\\ \text{smf}\end{Bmatrix} \left[\mid \begin{Bmatrix}\text{text-expr}\\ \text{smf}\end{Bmatrix}\right] \ldots, \begin{Bmatrix}\text{label}\\ *\end{Bmatrix}$$

Action:

If at least one of the operands matches the ANSWER register then the program branches to *label* (or takes the default branch if *). Otherwise, the next statement is executed.

Examples:

```
MATCH 'Identifier'|JOE|'Variable', L5
MATCH PEN(3)|PEN(5), L7
MATCH PEN(I), RESP(I)
MATCH PAT('¢AB¢'),*
```

b. UNREC-statement

Syntax:

$$\text{UNREC} \quad \begin{Bmatrix}\text{label}\\ *\end{Bmatrix} \left[, \begin{Bmatrix}\text{label}\\ *\end{Bmatrix}\right] \ldots$$

Action:

The i^{th} unrecognized response to the controlling SHOW-statement will cause a branch to the i^{th} label in the UNREC label list.

Example:

```
UNREC  *, *, L3
```

c. Unconditional branch

Syntax:

$$\begin{Bmatrix}\text{GOTO}\\ \text{GO TO}\end{Bmatrix} \quad \begin{Bmatrix}\text{label}\\ *\end{Bmatrix}$$

Examples:

```
GOTO *
GOTO L4
GOTO ARITH(4)
```

d. IF-THEN-ELSE

Syntax:

$$\text{IF} \begin{Bmatrix}\text{comparison-expr}\\ \text{logical-expr}\end{Bmatrix} \text{THEN} \begin{Bmatrix}\text{statement}\\ \text{DO-group}\end{Bmatrix}$$

$$\left[\text{ELSE} \begin{Bmatrix}\text{statement}\\ \text{DO-group}\end{Bmatrix}\right]$$

Action:

The expression in the IF-clause is evaluated to give a truth value as result.

Case 1: ELSE-clause not present:

If the truth value is 1, the THEN-clause is executed and control passes to the statement following the IF-statement.

If the truth value is 0, the THEN-clause is not executed.

Example:

```
IF QCOUNT > 3 THEN X <- X + 1
```

Case 2: ELSE-clause present:

If the truth value is 1, the THEN-clause is executed and control skips the ELSE-clause and passes to the next statement.

If the truth value is 0, the THEN-clause is skipped and the ELSE-clause is executed.

Example:

```
IF NQN(2) > 3 THEN X <- X + 1
              ELSE X <- X - 1
```

(4) Assignment

Syntax:

$$\text{variable} \leftarrow \left\{ \begin{array}{l} \text{text-expr} \\ \text{slide-expr} \\ \text{arith-expr} \end{array} \right\}$$

Action:

The expression on the right hand side is evaluated and the result is assigned to the variable on the left hand side. No data conversion is done; the attributes of the variable on the right hand side must agree with the attribute of the right hand side result. Identifiers with the TC or SC attributes may not appear on the left hand side. Note that the system variables CASE, SQZ and PAUSE are set by assignment.

(5) FRAME-statement

Syntax:

frame-name: FRAME

where frame-name is an identifier.

Action:

Serves to define the beginning of a set of statements which constitute a frame. Use of the frame facility is optional. It is used to take advantage of default branching and default screen formatting.

(6) DO-group head

Syntax:

[DO-group-label:] DO

where DO-group-label is an identifier.

Action:

Serves to define the beginning of a DO-group.

(7) Procedure-statement

Syntax:

procedure-name: PROC [(parameter[,parameter]...)]

where procedure-name and parameter are identifiers.

Action:

Serves to begin a DIAL procedure definition and to define the procedure's parameter list.

(8) END-statement

Syntax:

$$\text{END} \quad \begin{Bmatrix} \text{procedure-name} \\ \text{frame-name} \\ \text{DO-group-label} \end{Bmatrix}$$

Action:

a. procedure-name

Serves both to end the definition of a procedure, and upon execution, to return program control to the calling point.

b. frame-name

Defines the end of a frame

c. DO-group

Defines the end of a DO-group.

(9) CALL-statement

Syntax:

```
CALL procedure-name [(argument[,argument]...)]
```

Action:

The procedure named in the statement is invoked with the arguments (if any) in the argument list. Execution resumes at the statement following the CALL-statement.

(10) ENDLESSON-statement

Syntax:

```
ENDLESSON [lesson-name]
```

Action:

The following system message is displayed:

```
END OF LESSON
DO YOU WISH TO GO ON TO THE NEXT LESSON?
TYPE YES OR )OFF
```

(11) RESUME-statement

Syntax:

RESUME

Action:

Defines a resume point in a lesson. Chapter 5, Section 6, defines the RESUME process.

(12) Naming-statement

Syntax:

$$\text{identifier} = \begin{Bmatrix} \text{text-const} \\ \text{slide-const} \end{Bmatrix}$$

Action:

Names a read-only constant and gives the TC or SC attribute to the identifier. Note that such identifiers cannot appear on the left hand side of an assignment statement.

Examples:

```
pint = 'Press INT to continue'
 MVT = 2705
```

(13) Repetition statements

Syntax:

$$\text{DO WHILE} \begin{Bmatrix} \text{comparison-exp} \\ \text{logical exp} \end{Bmatrix}$$

group of statements

ENDDO [do-while-label]

Action:

The group of statements so bracketed is repeatedly executed while the expression in the DO-WHILE clause remains true. The decision is made <u>before</u> each repetition.

Syntax:

```
        REPEAT
        ┌
        │group of statements
        └
        UNTIL ┌ comparison-expr ┐
              └ logical-expr    ┘
```

Action:

The group of statements so bracketed is repeatedly executed until the expression in the UNTIL clause becomes true. The decision is made after each repetition; thus the group will be executed at least once.

4.16.3 Text manipulation

The three operators

SUBSTR,
INDEX, and
LENGTH,

together with concatenation, form a workable set of primitives for text manipulation. The definitions of SUBSTR, INDEX and LENGTH in DIAL are exactly the definitions of the built-in functions of the same names in PL/I-F [IBM 1970b: Section G] and are summarized as

```
SUBSTR (text-expr, j [,k])
INDEX (text-expr, configuration-text-expr)
LENGTH (text-expr)
```

Examples of string expressions using these operators are:

```
A || B || SUBSTR (ANSWER, 1, 3)
SUBSTR (EXAMPL, INDEX(C,'b̸') + 1)
LENGTH (ANSWER)
SUBSTR (INFORM,4)||ANSWER||'what you meant
                             to type?'
```

4.16.4 Expressions

An expression is a representation of a value. A single constant or a variable is an expression. Combinations of constants and/or variables, along with operators and/or parentheses, are expressions. An expression that contains operators is an operational expression. The constants and variables of an operational expression are called operands.

The rules for the five classes of expressions in DIAL are as follows:

Class	Operators	Valid Operands	Result	Example
STRING	SUBSTR ||	TEXT variables TC text constants	TEXT	A||'Dog' SUBSTR(ANS,4)||X
SLIDE	+ -	SLIDE variables SC slide constants INTEGER variables INTEGER constants	SLIDE	mvthm + 1
ARITHMETIC	+ - *	INTEGER variables INTEGER constants	INTEGER	a + b A+B*C*(X-Y)
LOGICAL	& | ¬	expressions having truth values	truth value	(AA=1)&(BB=1) PAT('¢5A7¢')
COMPARISON	> >= < <= = ¬=	any expression other than logical	truth value	I > 4 ANSWER = A||B LENGTH(ANS)>=4 IDENT > 'DOG'

Operands combined by an operator to form an expression must have the same attribute.

4.16.5 System matching functions

The smf's are PEN and PAT; both of them

(1) operate on the student's response (the contents of the registers ANSWER or PEN(1),...,PEN(8), and

(2) return a truth value.

a. PEN

Syntax:

```
PEN(arith-expr)
```

Action:

The results of the arithmetic-expression must be one of the integers 1 through 8. PEN (i) returns the truth value 1 if the light pen hit occurs on the i^th^ multiple-choice entry.

Examples:

```
PEN(4)
PEN(I)
```

b. PAT

Syntax:

```
PAT(text-expr)
```

Action:

The text-expression defines a pattern and PAT returns the truth value of 1 if the pattern matches ANSWER.

The pattern is made up of a series of pattern elements separated by a cent symbol, the "don't care" symbol, where noise characters may appear. A match occurs if each of the pattern elements (in the order they appear in the pattern) occurs in ANSWER. The symbol ¢ cannot be in a pattern element.

Examples:

(1) To test if ANSWER contains the key-letters A, B, and C:

```
PAT('¢A¢B¢C¢')
```

Note there are three pattern elements.

(2) To test for a substring of ANSWER

```
EX <- '¢AND¢'
MATCH PAT(EX),L4
```

(3) To specify an answer of the form "NEXT: CALL P(ALPHA,BETA);" :

```
SHOW 'Give an example of a statement
     which invokes the subroutine P
     having two parameters'
MATCH PAT('¢CALL P¢(¢,¢)¢;¢'),L5
```

The pattern elements in this example are:

```
CALL P
(
,
)
;
```

PAT('') matches only the null string. PAT('¢') matches all strings. PAT('A¢B') will match 'AB'.

4.16.6 Light pen usage

Multiple-choice items as targets must obey the following rules

(1) each item must begin with the character *

(2) each item must occupy no more than one row

(3) there can be no more than eight targets.

Light pen hits are specified by the PEN smf. The system recognizes only one hit at a time.

4.16.7 Default actions

Item	Default
Attribute	TEXT
CASE	on
SQZ	on
PAUSE	2 seconds
Branching with *::-	
MATCH	"Right" is displayed in green, then branch to next frame.
UNREC	The following message is displayed in yellow. Your answer was not recognized. It may be wrong, or it may be right in content but wrong in form, spelling or punctuation. Examine your answer and try again.
GOTO	Branch to next frame.
QVALUE	1
AVALUE	11
RVALUE	13

4.16.8 Abbreviations

These abbreviations are accepted:

Word	Abbreviation
ANSWER	ANS
MATCH	M
SHOW	S
SHOWAS	SAS
UNREC	U

4.16.9 Restrictions

(1) Character set

The CC-30 character set, for the purposes of DIAL programming, is divided into ordinary characters (those which have significance in the language) and string characters (those which may only occur in character string constants).

Ordinary characters:

A,B,C,...,Z, a,b,c,...,z

0,1,...,9

& ' ( ) * + , - : ;

< = > ¬ | ¢

String characters:

! " $ % . / ? @ \ ⌈ ⌉ ⌊ ⌋

° _ ←

(2) Length of Character strings and DIAL statements

The CRT screen imposes the maximum length; since there are 800 characters displayable on the CRT, and the CAI System reserves the use of rows 18, 19, and 20 in author mode, the maximum character string length and statement length is 680.

(3) Length of identifiers

All identifiers, CAI System-wide, may be up to ten characters long.

(4) Reserved words

These words may not be used as identifiers:

ANS
ANSWER
CALL
CASE
DCL
DO
ELSE
END
ENDDO
ENDLESSON
FRAME
GLOBAL
GO
GOTO
IF
INDEX
INTEGER
LABEL
LENGTH
M
MATCH
NEW
PAT
PAUSE
PEN
PLISUB
PROC
REPEAT
RESUME
RMV
S
SAS
SHOW
SHOWAS
SLIDE
SQZ
SUBSTR
TEXT
THEN
TO
U
UNREC
UNTIL
WHILE

(5) Other

Item	Range
Integer	-32768 to 32767
PAUSE	0 to 120 seconds
Slide	
carousel range	1 to 100
slide range	1 to 80
Number of light pen targets	1 to 8
PAT pattern elements	1 to 16
PAT pattern element length	1 to 80

CHAPTER 5

THE OPERATIONAL ENVIRONMENT

5.1 The host computer system

Instead of using a computer dedicated to CAI, as most workers have done, this project planned to use the IBM System/360 at the Triangle Universities Computation Center (TUCC), of which UNC is a one-third owner. Although a dedicated machine may be appropriate for public-school use, the use of a general campus facility, with both its system and staff resources, is especially economical for colleges. The major difficulties with this approach occur during system development and are common to most projects which involve embedding a sophisticated sub-system, using essential but often privileged services, into a host system already serving a large community of users. The approach was explored during Phase I of the UNC project and found to be sufficiently feasible in terms of system debugging inconvenience and service received by the working system, to adopt the same approach in the Phase II system. Although debugging in a non-dedicated environment is considerably more difficult, the benefits in our case, in addition to the economic ones, were considered to be worth the price paid. These benefits result from the extended scope of a comprehensive operating system (OS/360 with the MVT - Multiprogramming with a Variable number of Tasks - option) and include:

(1) the availability of PL/I as the language for programming the CAI System

(2) the availability of other language processors

   a. the Conversational Programming System (CPS), possibly for extending the answer processing capability of DIAL

   b. the PL/I (F) level compiler for portions of the answer analysis of constructed responses as suggested in Chapter 9.

(3) comprehensive file handling capability used for:

   a. instructional program storage

   b. logging (data gathering) of student responses

   c. the File Management System of the CAI System.

(4) a well established teleprocessing environment The primary data communications programs for the CAI System (those supporting the multiplexed CC-30's) are in the CHAT System. However, secondary requirements, e.g., administrative

programs, data analysis programs and systems debugging programs, were met by the existing remote job entry facilities of TUCC.

The TUCC system is described in [Brooks, et al., 1968; Freeman, 1968; Freeman and Pearson, 1968] where both technical and organizational aspects are discussed. The hardware configuration during the early part of the CAI System development was:

- S/360 Model 75 CPU
  - Main Storage 1024K bytes
  - Large Capacity Storage 2048K bytes
  - Disk Units 3 x 2314
  - Drum Units 2 x 2301
  - Magnetic Tape Units 1 x 2401-II, 4 x 2402-II
  - Line Printer - 1403
  - Card Reader-Punch 2540
  - Communications Equipment
    - 2701 Data adapter for high speed lines to S/360's at Duke, NCSU and UNC
    - 2703 Transmission control with 48 ports for low speed terminals

During the summer of 1971, TUCC replaced its Model 75 by a System/370 Model 165 with 2048K bytes of main storage. Replacing UNC's Model 50 as its on-campus terminal to TUCC is the Model 75 formerly at TUCC. The CHAT System, under which the CAI System runs, was developed and operated on the Model 75 while it was at TUCC. It operates entirely from the low-cost 8 microsecond storage (LCS). The new Model 165 at TUCC does not have such storage; its two million bytes of fast storage are less than the former TUCC configuration. Therefore the CHAT System was moved to UNC with the Model 75, and it continues to operate from slow core.

The host computer system for production use of the CAI System is therefore a System/360 Model 75 with the following configuration.

S/360 Model 75 CPU

- Main Storage 256K bytes
- Large Capacity Storage 1024K bytes
- Disk Unit 2314
- Magnetic Tape Units 2 x 2415
- Line Printers 2 x 1403
- Card Reader-Punch 2540
- Card Reader-Punch 2540 } housed in IRSS building nearby
- Line Printer 1403 }
- Plotter
- Graphics Display System
  - Vector General and PDP-11/45
- Communications Equipment
  - 2701 Data adapter for high speed line to TUCC
  - 1270 (Memorex) Data adapter for low speed and medium speed lines

CC-7012 (CCI) Channel adapter

5.2 The Chapel Hill Alphanumeric Terminal (CHAT) System

The CHAT System was designed and implemented by Gary D. Schultz [1973] of the UNC CAI Project. It is a single-region[1] resident time-sharing system wherein various programs share the execution time by CHAT's use of the OS/MVT multitasking facilities. An application program runs as a subtask with respect to one of the executive tasks of CHAT. This monitor program also provides the CC-30 input/output programming support to application programs.

While CHAT was operating at TUCC, a medium speed communication line (2400 bits per second) connected the CAI Center to TUCC. The proximity of the CAI Center to the UNC Computation Center installation has made it possible to dispense with the communications line and use a direct hardwire connection instead. The CHAT System hardware configuration at UNC is shown in Figure 5.1. The line transmission speed with this direct connection causes the CRT screen to be filled in one-tenth of a

---

[1] The main memory subdivision that is allocated to a job step in OS/360 is known as a region under the MVT option and a partition under MFT.

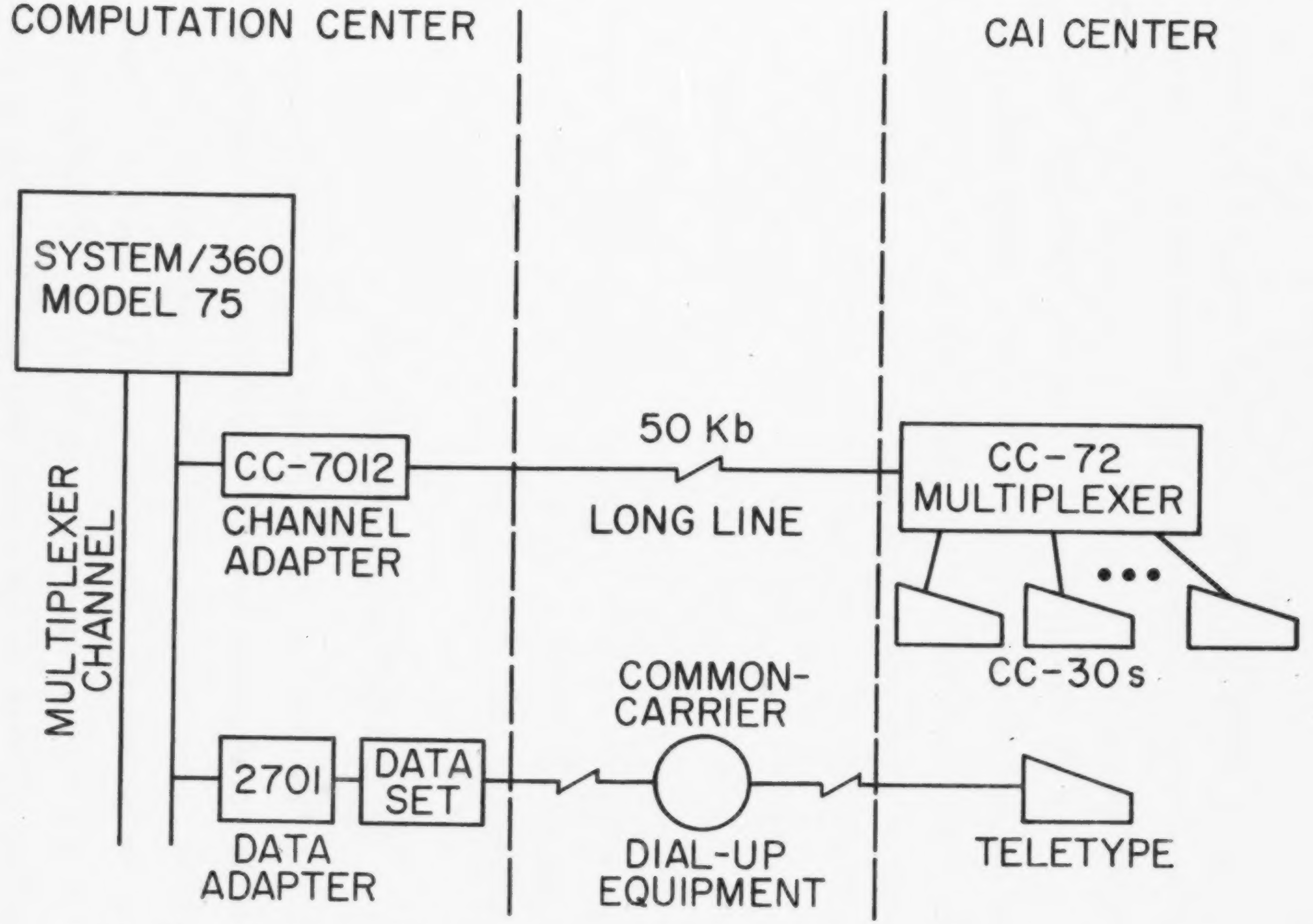

COMPUTATION CENTER
CAI CENTER
SYSTEM/360 MODEL 75
MULTIPLEXER CHANNEL
CC-7012
CHANNEL ADAPTER
2701
DATA SET
DATA ADAPTER
50 Kb
LONG LINE
COMMON-CARRIER
DIAL-UP EQUIPMENT
CC-72 MULTIPLEXER
CC-30s
TELETYPE

Figure 5.1 -- The CHAT System hardware configuration.

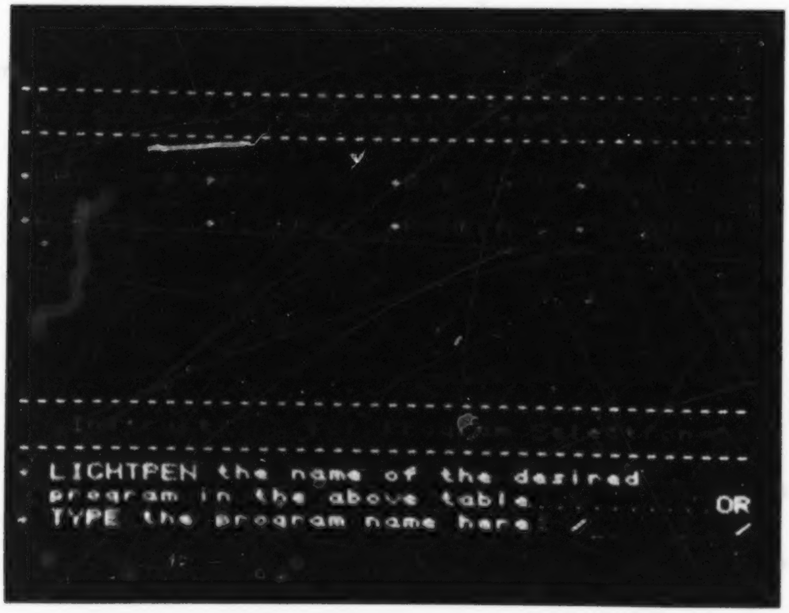
second. Using the former 2400 bit line, we experienced a 2 1/3 second screen-fill time, which we found to be quite adequate.

The UNC CAI System is a complete subsystem running as an application program under CHAT. Other subsystems include Brown University's Hypertext editor, a numerical analysis laboratory simulator and an interactive assembler.

Access to a subsystem of CHAT is provided by the CHAT Monitor Table of Contents (MTOC) display shown in Figure 5.2. The default selection is CAI, i.e., depressing INT in response to MTOC's invitation is equivalent to either pointing to CAI or typing CAI and depressing INT.

5.3 The student/author work station

5.3.1 The work station, pictured in the frontispiece, is designed around a Computer Communications Incorporated CC-30 Communications Station. The work station is normally used by just one person (a student or author) but has been designed also to allow two students to be seated so that we can experiment with learning in pairs, with perhaps one student using the keyboard and the other the light pen.



```
- LIGHTPEN the name of the desired
  program in the above table.          OR
- TYPE the program name here: /_          /
```

Figure 5.2 -- The CHAT Monitor Table of Contents (MTOC) display.

Output to the user is displayed either on the CRT or on the slide screen above the CRT. A further output facility, audio tape, is being studied for later inclusion.

Input from the screen is either a message typed from the keyboard and displayed on the CRT, or a light pen position on the CRT.

A desk work area is part of the work station; authors can use it for course-plan notes, instructional program listings, etc.; students for note-taking and performing exercises which require pen and paper.

A proctor call switch is mounted on the left side of the display housing. When switched, a buzzer sounds in the proctor office and panel lights show which station is calling.

5.3.2 The CCI CC-30 Communications Station

The nucleus of the station is the CC-301 TV Display Controller. This has three major sections: a magnetic core buffer memory, a character/graph generator, and an input-output section. The buffer has two functions: it is both the data source for refreshing the CRT, at the rate of sixty times per second, and the storage facility for the station.

The CC-300 TV Display is a standard television set or television monitor. In the alphanumeric mode the characters stored in the buffer memory in ASCII format are displayed on the CRT in a format of 20 lines of 40 characters each. When operating in graph mode, data are displayed by means of a 108 x 85 matrix of dots.

The CC-304 Light Pen, similar in shape, size and weight to an ordinary fountain pen, employs a phototransistor detector. When it is directed towards the display, a marker appears on the CRT indicating the character position at which the light pen is positioned. This marker is a brightening of the character background. The coordinates of the character are stored in the CC-301 when the interrupt button on the pen is depressed.

A standard Model 850 Kodak Random Access Carousel slide projector is connected as an output unit to the CC-301 by a specially designed interface. The four commands for the projector are: lamp on, lamp off, show slide nn and show the next slide.

Further details on the station are given in the manufacturer's manual [CCI, 1968].

5.3.3 Gaining access to the CAI System

As soon as the user is seated at the work station he depresses the INT key, which results in the CHAT Monitor Table of Contents display[2] of Figure 5.2. Depressing INT once more initiates the CAI System which then invites the user to sign-on, by displaying the message

```
SIGN ON BY ENTERING YOUR ID
)SIGN ON _
```

Whether he is an author or student is determined by his identification number. An author is put under the control of the program routine AUTHOR and invited to enter his first command. A student identification number causes the appropriate instructional program to be loaded and execution of it to begin, after a restart procedure if necessary.

5.4 System overview

This section is intended to provide a background to the discussion, in subsequent sections, of author and student use of the system.

The overall flowchart of Figure 5.3 shows the two main parts of the CAI System, student and author modes.

---

[2] This is the only direct contact (c.f. Figure 4.6) that a CAI System author/student user has with CHAT under normal operating conditions.

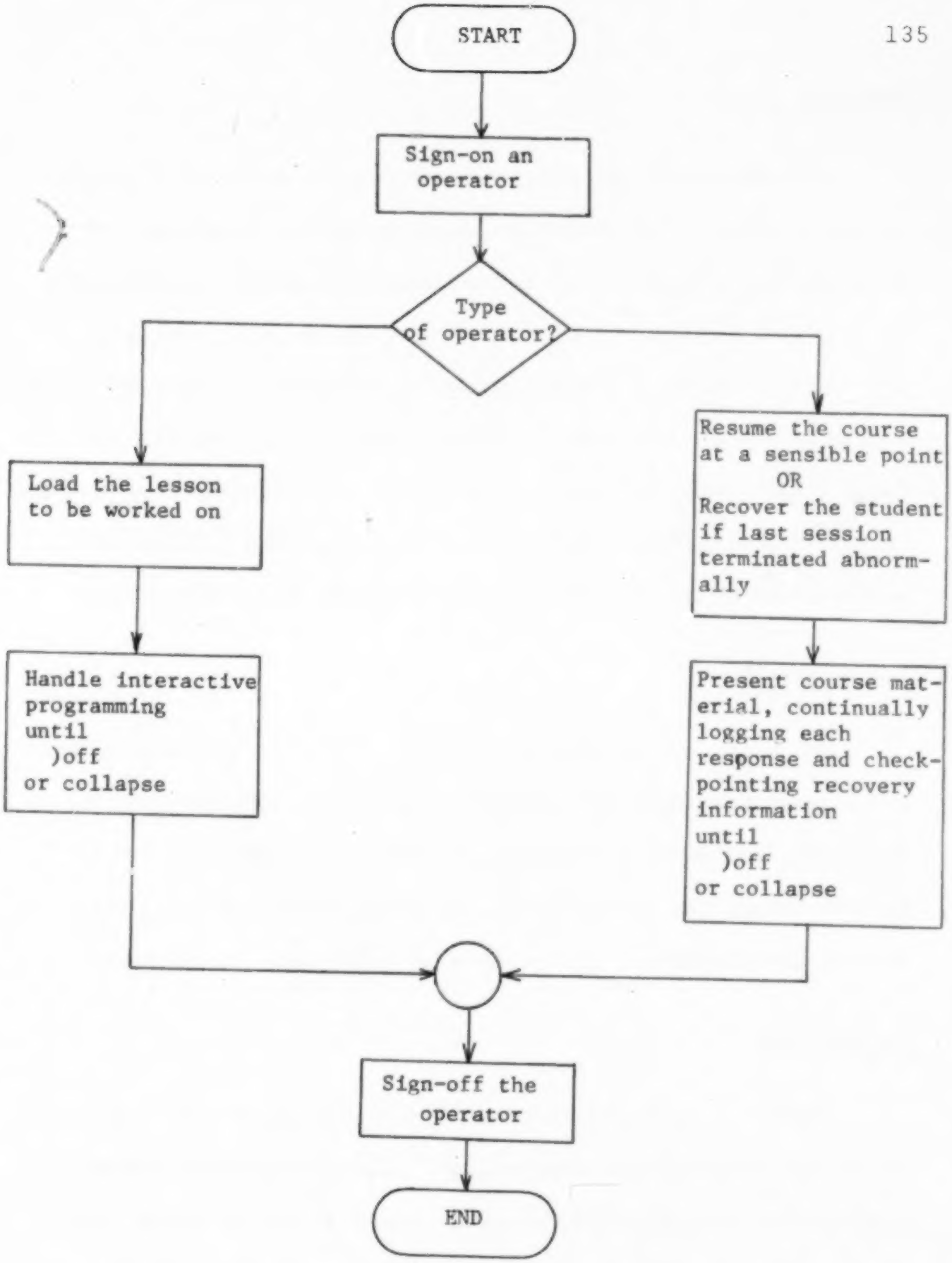

START
Sign-on an operator
Type of operator?
Load the lesson to be worked on
Handle interactive programming until)off or collapse
Resume the course at a sensible point OR Recover the student if last session terminated abnormally
Present course material, continually logging each response and checkpointing recovery information until)off or collapse
Sign-off the operator
END

Figure 5.3 -- An overall flowchart of the CAI System.

Student mode

Presentation of course material to a student occurs in this mode. The instructional programs prepared by authors are executed in a paging environment implemented in the CAI System. The run-time storage environment for each student's execution of a program is carried from one session to the next. Each action of a student is logged for later off-line analysis, and status information is continually checkpointed to minimize the effects of system breakdown on students' progress and attitudes.

Author mode

Preparation of course material occurs in this mode. A command language interpreter controls the compiler for DIAL and author testing of program segments. The system tries to anticipate, at every step during interactive programming, the author's next type of input.

Protection

There is absolute protection of a lesson in use by students from author tampering. Identification number protection is provided between users (both authors and students). Protection against UNC Computation Center system failure, CAI equipment failure and CAI software failure is attempted. Additionally, the operator in student mode is protected against making any change in course material or any explicit change in other files.

Course structure

A set of lessons constitutes a course; the only communication between lessons is by means of identifiers with the GLOBAL attribute. Such a course structure was designed to meet the following requirements

(1) flexibility in course preparation;

(2) the setting of a practical (from an implementation view) maximum program size;

(3) protection of sections of course material in student use from author tampering.

File management system

With the exception of the student record and author record files, which are held on two OS/360 ISAM (Indexed Sequential Access Method) data sets, all logical files for the CAI System (files for log, instructions, source code, recovery, directories, etc.) are held on one physical OS/360 BDAM (Basic Direct Access Method) dataset named CAIFILES. The File Management System, the part of the system which handles the management of CAIFILES, is not described in the thesis since it is not seen by the user, but is covered elsewhere in the system documentation [Mudge, 1972]. Briefly, its functions are to handle

(1) disk storage management;
(2) the various logical files;
(3) the coordinated use of external serially reusable resources;
(4) directories of courses, lessons, source code, etc.

5.5 Instructional programming in DIAL - author use of the system

Since Chapter 4 gives the DIAL specification, we are here concerned only with the command language, i.e., we treat the mechanisms for interactively programming in DIAL.

Each command is preceded by the character), e.g.,)list, chosen because no syntactically valid construct in the language can begin with closing parenthesis. The author converses with AUTHOR by means of the command language and the statement-numbering mechanism.

Before a DIAL statement can be entered, the lesson to which it belongs must be defined. If a new lesson is being created, the)lesson command is entered, in the format

```
)lesson lesson-name
```

and causes the appropriate directory entries to be made and disk storage space allocated. If a DIAL statement being entered is to be a change or an addition to an existing lesson, then that lesson is defined by the)load command.

A statement number must appear to the left of each DIAL statement. It provides the author and the system with a way of referring to the statement which follows it. Consequently, every statement number must be unique; if two statements are entered with the same number, only the one typed last will be retained. Numbers must lie between 1 and 9999.[3] The author can ask the system to generate statement numbers by preceding any line with)m,n where m is the base number and n is the desired increment. The system signifies its acceptance of a valid numbering request by overwriting the request with the first statement number. For example,

```
)200,10
```

would be accepted and overwritten by 200. Then, after a DIAL statement has been accepted, the system would prompt the author by displaying 210.

---

[3] A numbering scheme allowing decimal points was rejected to conserve precious CRT space.

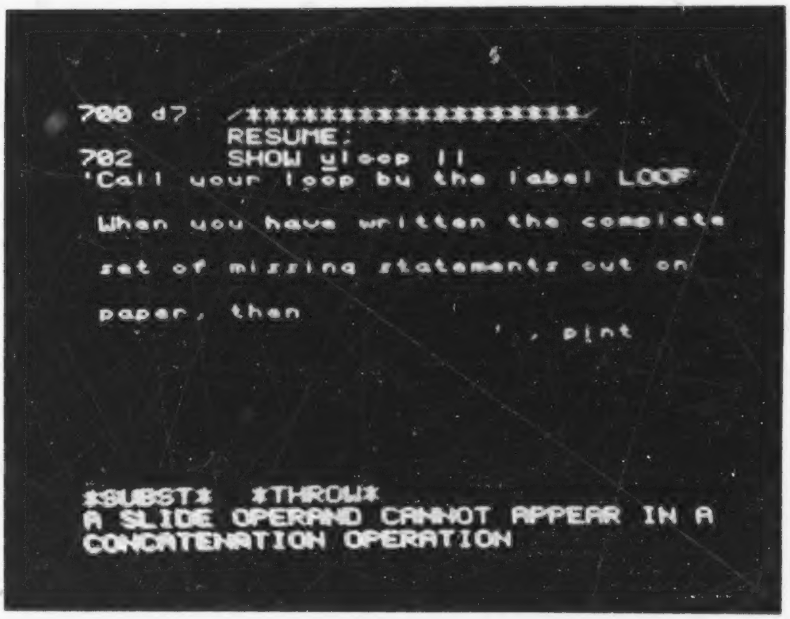
Although statements in a DIAL program may be entered in any order, their order for execution is determined by their numbers.

The role of the statement-numbering mechanisms is thus twofold:

(1) it serves to indicate that code entered by an author has been accepted as error-free by the compiler. This is signalled by the system displaying the next statement number and enabling the keyboard.

(2) as an editing facility: statements may be replaced, deleted or inserted by preceding a statement with the appropriate statement number.

The CRT screen format is shown in Figure 5.4. The first seventeen lines of the screen are available to an author for entering statements. Line 18 displays the light pen buttons, and lines 19 and 20 are used to display diagnostics given by the compiler or command language interpreter. The figure shows a typical diagnostic, with the cursor at the position at which the compiler thinks the error has occurred. When an author has used down through line 17 of the screen, the CAI System clears the screen and resumes at line 1. This action is known as



```
700 d7  /********************/
        RESUME:
702     SHOW uloop ||
'Call your loop by the label LOOP

 When you have written the complete

 set of missing statements out on

 paper, then
                            ' , pint




*SUBST*  *THROW*
A SLIDE OPERAND CANNOT APPEAR IN A
CONCATENATION OPERATION
```

Figure 5.4 -- The cathode-ray tube screen format in author mode.

throwing and can be forced by pointing at the light pen function *THROW*.

To view the execution of a segment of his lesson, an author types)xeq m,n, where m and n are the statement numbers delimiting the segment. The other options for the)xeq command are:

)xeq m	begin execution at m and end at the highest number known in the lesson
)xeq m.	execute statement m only
)xeq	begin execution at the lowest statement number

The parameters m and n can be DIAL labels as well as statement numbers.

Execution of a program segment can be terminated by the author entering)stop. Of course the keyboard must be enabled for him to do this, and it will be so whenever his program is expecting a student input.

When an author has fully tested a lesson he attaches it to a course by the command

```
)attach  lesson-name  to  course-name
```

The lesson then becomes inaccessible to him in author mode. If he wants to keep an accessible copy for himself, he can do so by using the command

```
)rename new-lesson-name
```

which will make a copy of the lesson currently loaded and name the copy new-lesson-name.

A group of one or more DIAL statements may be retrieved from a library and included in a DIAL lesson by the)include command. Two libraries are available to an author, his own and the public library to which all authors have access. The)include command has two forms

```
              )include group-name
and           )include group-name public
```

There are no structural restrictions on the DIAL statements which constitute a group, e.g., the group need not be a subroutine. A group is put into a library by

```
)group group-name
   .
   .
   .
   .
 the group of DIAL statements
   .
   .
   .
   .
)endgroup
```

The form

```
                )group group-name public
```

is used for the public library.

As an example, consider a set of character constants an author would like to use in each lesson in a course he is building but wishes to avoid entering them for each new lesson. The group would be placed in the library by

```
     )GROUP CCONS
     100 /*character constants:-*/
     110 /***********************/
     120 OBS='Observe the slide above '
     130 PINT='PRESS INT TO CONTINUE'
     140 TYNO='Type yes or no'
     150
     )endgroup
and  )inclusion followed by a )listing
     900
     910
     )INCLUDE CCONS
     )LIST 900
```

would appear as

```
     900
     910
     920 /*character constants:-*/
     930 /***********************/
     940 OBS='Observe the slide above'
     950 PINT='PRESS INT TO CONTINUE'
     960 TYNO='Type yes or no'
     970
```

The remaining commands and their functions are:

)load lesson-name	locates the named DIAL lesson in the author's directory and by loading it makes it available for him to work on.
)list m,n	displays the program segment delimited by statement numbers m and n. If more than seventeen screen lines are contained in the segment, it is shown in successive batches of 17, the author pressing INT to obtain the next batch.

	Thus the author can quickly "page through" a DIAL lesson. The options)list m,)list m. and)list are available and the scopes are the same as for)xeq.
)delete m,n	deletes the program segment delimited by m and n. The options of)xeq are available.
)reseq m, n from p by q	Insertions and corrections often make the the final form of a program considerably different from that of its early stages. To avoid the inconvenience that this can cause, e.g., in trying to insert a statement between 1004 and 1005, the)reseq command is provided. The command resequences the program segment delimited by m and n beginning with p and using q as increment.
)directory	lists all lessons in the author's directory.
)number	The command is overwritten with the next free statement number for the lesson loaded.

)purge lesson-name	Purges the lesson from the system. Since the actions performed by this command are irrevocable, a response is sent instructing an author to repeat the command; then, if he does, the purging is carried out.
)print [lesson-name]	Produces a printed listing on the host computer. If no lesson-name is given, the lesson currently loaded is printed.
)lid	The command is overwritten with the name (ID) of the lesson currently loaded.
)off	Invokes the sign-off procedure and hence termination of the session.

A summary of the commands is given in Figure 5.5. Three-letter abbreviations for each of the commands are acceptable. The commands may be entered in upper or lower case.

Reading from the screen

Whenever the keyboard is enabled, an author is free to move the cursor to any of the 800 character positions. But the CHAT interface uses the cursor position to define

AUTHOR COMMANDS

Function	Command	Other Options
sign off	)off	
new lesson	)lesson <>	
listing	)list	)list m
		)list m.
		)list m,n
	)print	
execution	)xeq	)xeq m
		)xeq m.
		)xeq m,n
	)stop	
general	)load <>	
	)lid	
	)m,n	
	)number	
lesson/course structuring	)directory	
	)rename <>	
	)attach <> to <>	
	)purge <>	
editing	)delete	)delete m
		)delete m.
		)delete m,n
	)reseq i,j from k by ℓ	
library	)include <>	
	)group	
	)endgroup	

Notes:
1. <> is a lesson or course name.
2. m and n are statement numbers or labels.
3. i,j,k, and ℓ are statement numbers.
4. Three letter abbreviations of the commands are acceptable.
5. Either upper or lower case can be used for the commands.

Figure 5.5 -- The summary sheet of commands for work station use.

which part of the screen will be transmitted to the CAI System as author input. Hence it was necessary to establish a convention for reading from the screen.

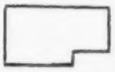
A window is that part of the CRT which the CAI System will read. Changes made or new text entered by an author will be effective only in this window.

The system expects the window to contain either a DIAL statement or a command and will respond with a diagnostic message otherwise.

After processing the contents of a window, the system tries to anticipate the type of the next action and positions the cursor in the row of the new window top. The end of the window is the position of the cursor when the author depresses INT to send his action to the system.

The window may be moved by an author at any time, for example, to cover a DIAL statement further up the screen, by pointing the light pen to the row he wants to be the new window top.

To summarize, screen reading is defined by a window

(1) of shape  ,

(2) of variable size (1 to 17 rows) according to the length of the author action it contains,

(3) with its top being the row where the cursor was left by the system (anticipated next move or in response to a move window request),

(4) with its end being the cursor position when INT is depressed.

When DIAL statements are being entered sequentially the window top moves down after each statement has been accepted. During editing an author will move the window to cover a statement in a segment he has)listed.

The action of the light pen function *THROW* can be recast in terms of the window - a throw request causes the screen to be cleared and the window, with its current contents, to be repositioned with its top at row 1.

The *SUBST* function

A text manipulation facility is available to an author, with which he uses the light pen to point to parts of his program at which he wishes to substitute, insert, or delete text. To use this facility he first points at *SUBST* with the light pen. The system then displays the prompt TEXT? on row 18 (overwriting SUBST in green) and positions the cursor at the beginning of row 19 inviting him to enter the text. When he has entered the text, he presses INT, is asked FROM? and points to a position on the screen; then he is asked TO? The system inserts the text, restores the *SUBST* button, and enables the keyboard again.

The *SUBST* function can also be used to delete text if a null string is entered. In summary, the protocol for *SUBST* is

(1) pen *SUBST*

(2) TEXT? - if author enters nothing, then a delete function occurs,

- if author enters non-null, then a substitute or insert function occurs,

(3) FROM? - start of replaced text,

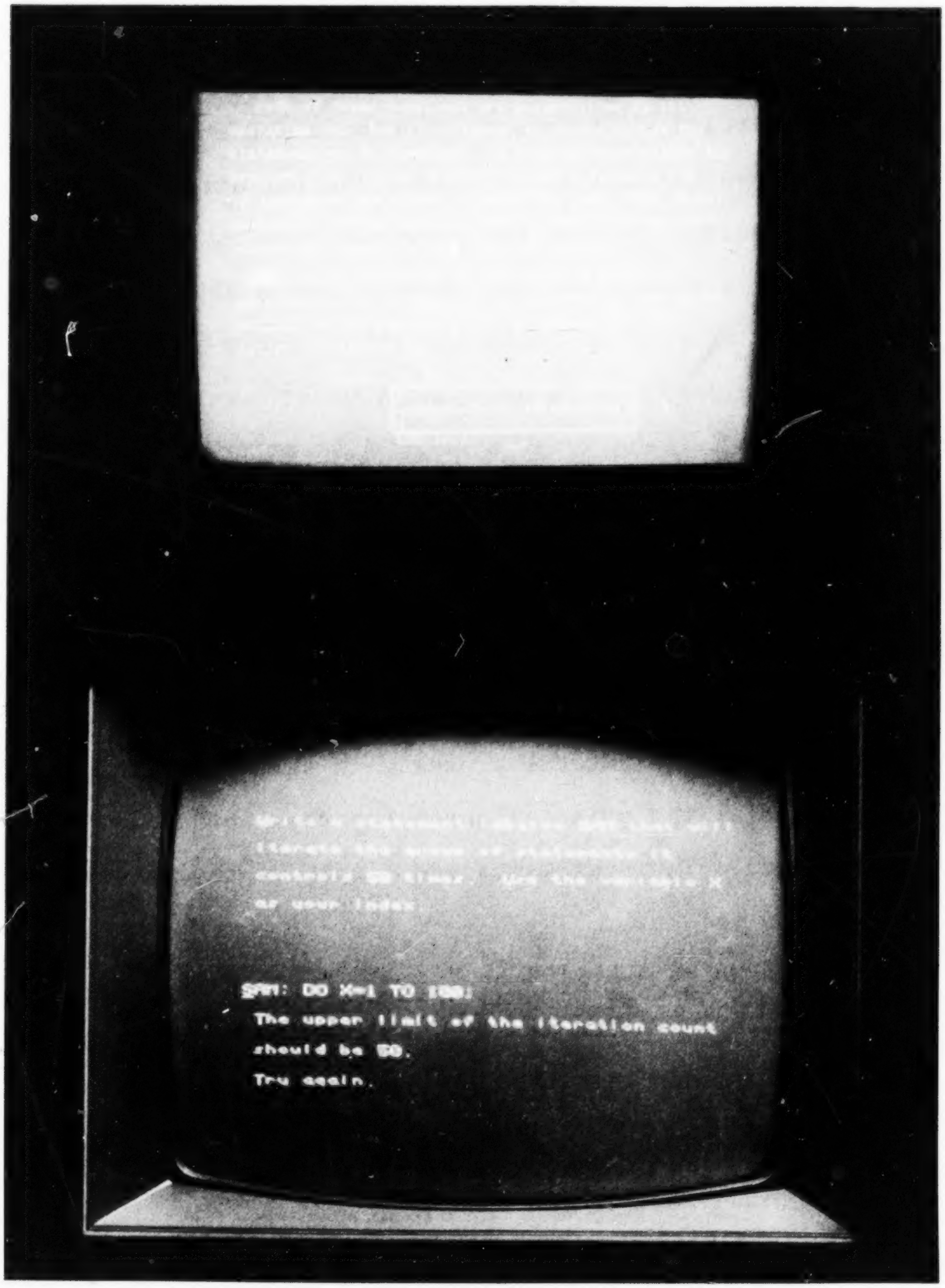
(4) TO? - end of replaced text.

5.6 The execution of an instructional program - student use of the system

Because the current design is strictly author-controlled CAI, there is only one system command available to the student, namely

```
)off
```

All other responses from the student are elicited by the author, and take the form of typed input or light pen input. Figure 5.6 shows the CRT and slide displays during a typical student interaction.



```
SAM: DO X=1 TO 100;
 The upper limit of the iteration count
 should be 50.
 Try again.
```

Figure 5.6 -- The cathode-ray tube and slide displays during a typical student session.

The point in a lesson at which a student is restarted at each new session has received special attention in the system design. Unless the previous session terminated abnormally, the system goes through the RESUME process in which the student is restarted at an author-specified point in a lesson. To help the student's orientation, this point is usually at a frame just prior to the end of the last session. The system message displayed to indicate a RESUME is

```
NORMAL RESUMPTION -- YOU ARE RESUMING
LESSON c AT A SUITABLE POINT
PRIOR TO WHERE YOU SIGNED OFF.
```

If the previous session terminated abnormally, the system goes through the RECOVER process, in which the student is restarted at the point at which recovery information was last taken. The system message displayed to indicate a RECOVER is

```
ABNORMAL RESUMPTION -- YOU ARE RESUMING
IN LESSON b AT A POINT JUST PRIOR
TO SYSTEM FAILURE.
```

Thus during each session the system needs to periodically copy RESUME and RECOVER information to CAIFILES. At sign-on, to choose between the two processes, the system uses the setting of the switch RECOVNEEDED on the student record file STUREC. The logic of its settings is given in Figure 5.7.

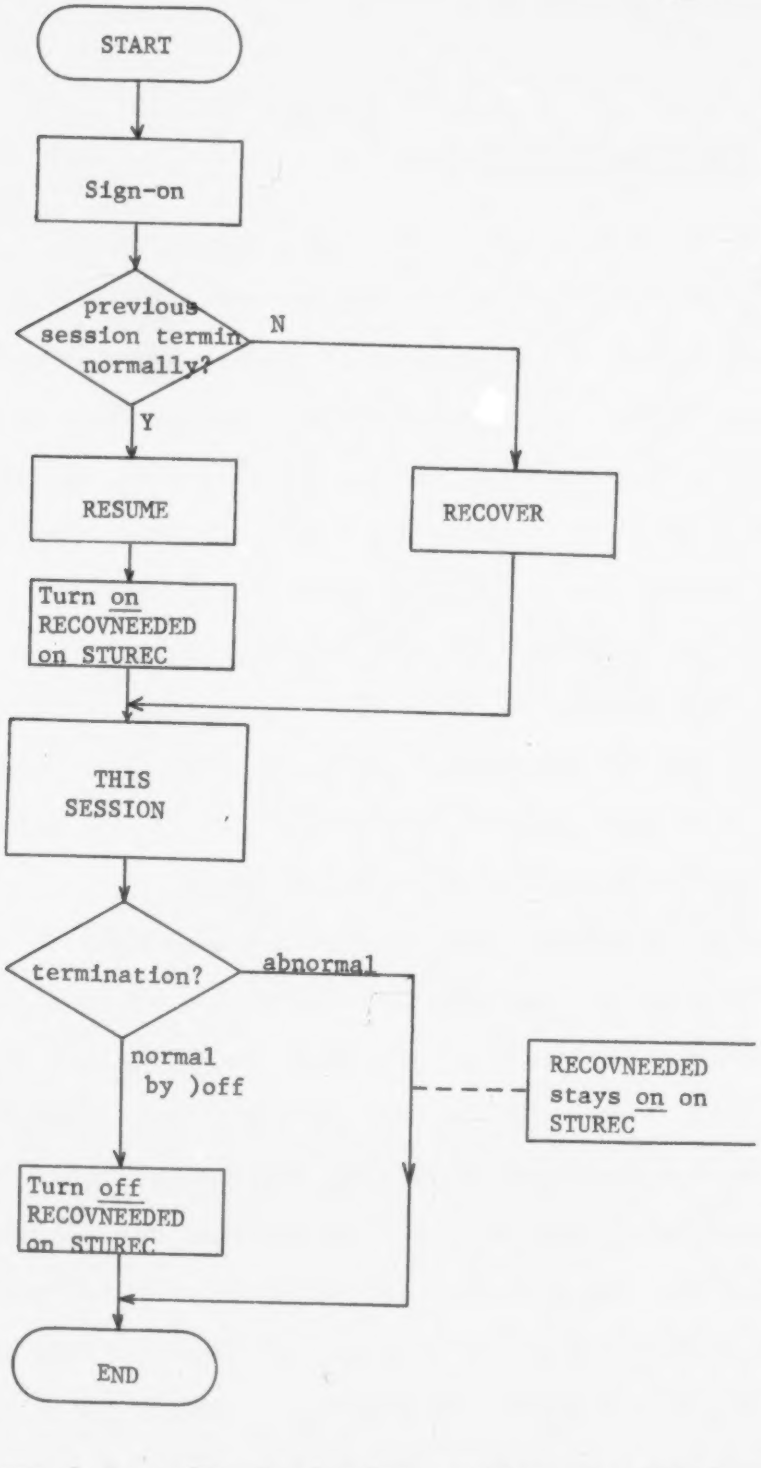

START
Sign-on
previous session termin normally?
N
Y
RESUME
RECOVER
Turn on RECOVNEEDED on STUREC
THIS SESSION
termination?
abnormal
normal by)off
RECOVNEEDED stays on on STUREC
Turn off RECOVNEEDED on STUREC
END

Figure 5.7 -- The logic for setting the RECOVNEEDED switch.

5.7 Proctor facilities

A third type of user of the system is the proctor, who is present in the CAI Center building during all student sessions, and has both administrative and pedagogical roles. To date we have examined only his administrative duties, where, for example, he is responsible for introducing new students to the system (by entering the student's identification number on to the student record file and teaching him to operate the work station) and dealing with operational difficulties during student sessions. In his pedagogical role the proctor is the on-site representative of an author; this thesis does not examine this role.

The proctor terminal is a standard Type 33 ASR Teletype located in the proctor office. A current implementation restriction that prevents communication between sub-tasks in the CAI region, in particular between the proctor sub-task and a student sub-task, consequently prevents real-time interaction between the proctor and the student via the system. Interaction of this type could, for example, cause the proctor terminal to type a message that a given student had reached a RESUME point in the instructional program. Abnormal conditions such as student station failure or the student entering proctor mode, could also be signaled.

5.7.1 Proctor override

A facility is provided whereby the proctor can override the normal sequence of execution of an instructional program. Our Phase I experience clearly revealed the need for this facility

(1) to correct situations caused by hardware and software failures,

and (2) to enable the student to repeat a particular segment of the course.[4]

Although the RESUME and RECOVER processes should handle most of the problems, I feel there is still a need for an override mechanism.

The proctor can use this mechanism at any stage in a student session after sign-on by entering)proctor on the display. The system responds with)_ and the proctor then completes

```
)override statement-number
```

where statement-number is the DIAL source-code statement in the current lesson at which he wants program execution to be resumed.)override lesson-name causes a jump to the beginning of the named lesson.

---

[4] This situation is intimately involved with the instructional process and, like other forms of pedagogical assistance given by the proctor, must be strictly controlled in any experiment aimed at evaluation of CAI.

5.7.2 Administrative programs for proctor use

These off-line programs, most of which were written by Robert Cannon, have facilities for

file maintenance of STUREC, the file of student records,
file maintenance of AUTHREC, the file of author records,
printing the log file held on CAIFILES,
and reporting the student statistics gathered by the system.

An on-line file inquiry program has been written by Mitchell Bassman to display the contents of a student's STUREC record. The program is used at a student/author work station but is accessible only to proctors.

CHAPTER 6

MODIFYING AND EXTENDING DIAL - THE TRANSLATOR WRITING SYSTEM

6.1 Introduction

An important requirement in the design of DIAL was to include in its implementation the ability to modify and extend the language. The purpose of this chapter is to give an understanding of the Translator Writing System to a level which will be helpful in assessing the flexibility of the implementation of DIAL. Furthermore, the chapter should provide a perspective for a systems programmer working with an author in changing the language.

Translators for high level languages are among the most complex types of computer programs and hence are expensive to build. Research in computer science, in particular in formal language theory, and experience with existing translators have led to a better theoretical and practical understanding of the processes involved in writing translators. Translator Writing Systems are now available which automate major portions of the task. For an excellent review of the field, see [Feldman and Gries, 1968].

The utility of a Translator Writing System (TWS) is based on the observation that most translators have many tasks in common - scanning of source text, syntax analysis, and generation of output. If these problems are solved once, in general form, the writer of an individual translator can concentrate on that part of the job which is unique to his translator, i.e., the connection of his meanings (semantics) to his forms (syntax).

The TWS built for the CAI System to enable users to modify and extend DIAL is based extensively on the TWS designed by McKeeman and others at Stanford University. This latter system and the construction of a translator for the language XPL are described in [McKeeman, et al., 1970]. In this thesis it is referred to as the XPL TWS.[1]

The metalanguage BNF (Backus-Naur Form) is used to describe the syntax of a language for which a translator is to be built. The semantic definition of the language consists of a set of user-written routines in the completed translator, where a semantic routine is called each time a syntactic construct has been recognized in the source language program being translated.

---

[1] For a complete treatment of certain topics, I direct the reader to [McKeeman, et al., 1970]. In this chapter it is referred to as A Compiler Generator.

The TWS in the CAI System (CAI TWS) consists of two principal parts:

(1) CONSTRUCTOR,[2] a translator from BNF into syntax tables (also called recognition tables) used in:

(2) COMPILER, a table-driven translator written in PL/I using user-written semantic routines.

These two parts are shown in Figure 6.1.

6.2 The compiler and the CAI System

Each DIAL compiler is produced with the aid of the CAI TWS and is known as the routine COMPILER in the CAI System. The routine named AUTHOR converses with an author, and, in response to a DIAL statement, invokes COMPILER as shown in Figure 6.2. The object code generated is a set of machine language instructions for a conceptual machine, called a delta machine. The instruction format is one-address. An interpreter for delta code is in the routine named EXECUTOR. This routine executes programs for both student and author modes.

---

[2] This part is called ANALYZER in A Compiler Generator. I prefer the term constructor from [Feldman and Gries, 1968] to avoid confusion with the term (syntax) analysis in compiler construction.

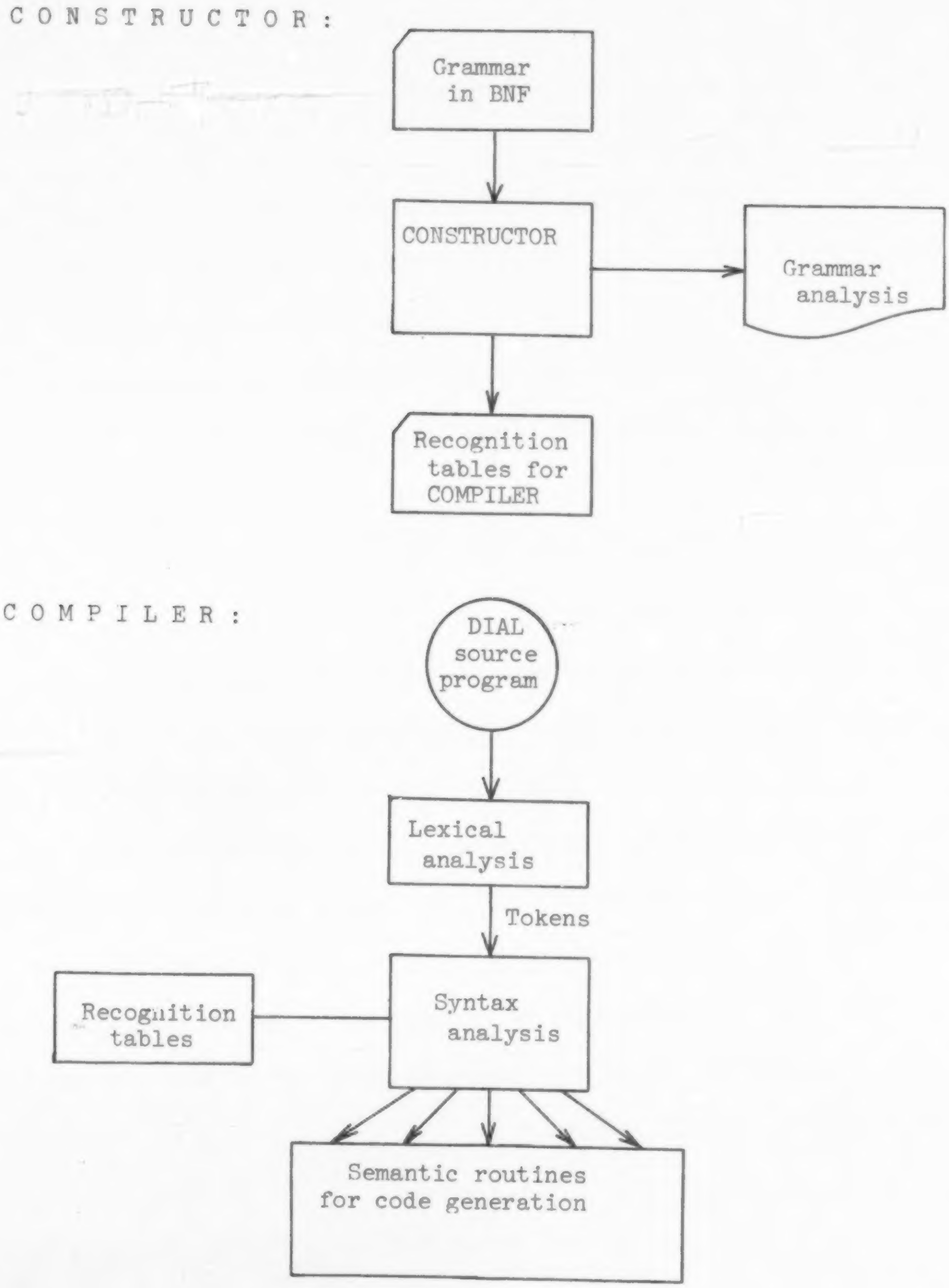

CONSTRUCTOR:
Grammar in BNF
CONSTRUCTOR
Grammar analysis
Recognition tables for COMPILER
COMPILER:
DIAL source program
Lexical analysis
Tokens
Recognition tables
Syntax analysis
Semantic routines for code generation

Figure 6.1 -- The two parts of the translator writing system.

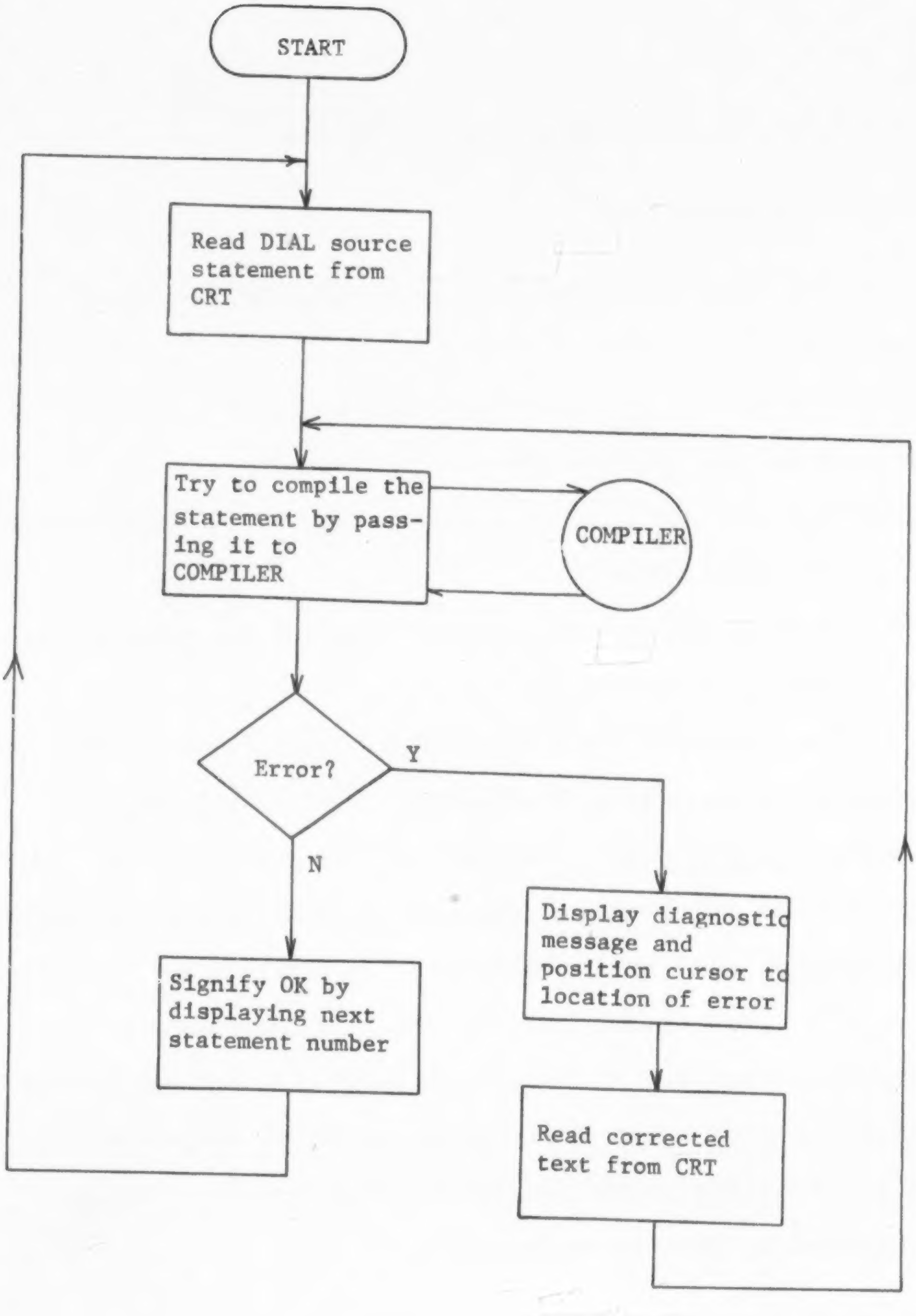

START
Read DIAL source statement from CRT
Try to compile the statement by passing it to COMPILER
COMPILER
Error?
Y
N
Signify OK by displaying next statement number
Display diagnostic message and position cursor to location of error
Read corrected text from CRT

Figure 6.2 -- The invocation of COMPILER by the controlling routine AUTHOR.

6.3 The CAI translator writing system

6.3.1 Introduction

The CAI TWS differs from the XPL TWS in three respects.[3]

(1) Due to the lexical flowgraph notation, lexical analysis is less grammar dependent.

(2) The CAI TWS uses PL/I, not XPL, for the description of translators.

(3) Some *a priori* knowledge of the CAI language environment has been used.

To elaborate this third point, consider further the generality requirement of a TWS. Syntax analysis is grammar *independent* (provided, of course, that the grammar is acceptable), code generation is highly grammar *dependent* and lexical analysis is normally grammar dependent. As regards lexical analysis we know, for example, that the CC-30 character set is constant, and that the wide variety of tokens normally encountered, e.g., floating point constants, will probably not be required in the CAI environment.[4]

---

[3] Since these are only minor differences, *A Compiler Generator* is still the major documentation for the CAI TWS.

[4] Except, of course, as data for DIAL programs.

This *a priori* knowledge of the tokens to be encountered has reduced the extent to which SCAN must allow easy modification. SCAN can therefore take a reasonably simple approach to lexical analysis and yet, by the lexical flowgraph notation, be fairly independent of grammar changes within the DIAL environment.

For some grammatical constructs there is a choice (in the construction of any compiler) between recognizing them in the syntax analysis phase and in the lexical phase. As a policy (aimed at keeping COMPILER as systematic as possible) the burden of all but trivial recognition tasks is put onto the syntax analyzer. This generality and systematization is achieved at the expense of efficiency, but seems worth the cost.

Note that the recognition of an identifier, which has the following syntactic definition,[5] is done by SCAN.

```
<identifier> ::= <letter>
               |<identifier><letter>
               |<identifier><digit>
<letter> ::= A|B|.... |Z|a|b ......|z
<digit>  ::= 0|1|...|9
```

---

[5] The length restriction (ten) is not shown in this BNF definition.

Since this definition is unlikely to change as DIAL is extended, the TWS loses little in generality.

6.3.2 CONSTRUCTOR

The output of CONSTRUCTOR is a set of recognition tables in the form of PL/I DECLARE statements (with the INITIAL attribute used to provide table values) for the version of COMPILER being built. This PL/I version of the constructor was supplied by John Walters [1970].

An XPL constructor is used for most of the runs during debugging of the grammar expressed in BNF as it is much faster and produces a better grammar analysis. However, it produces XPL declarations. Therefore the final run of a set of BNF is done on the PL/I version.

CONSTRUCTOR is described in *A Compiler Generator*, Chapters 7 and 10.

6.3.3 COMPILER

COMPILER is made up of three main parts:

(1) SCAN, which performs the lexical analysis and passes tokens to:

(2) ANALYZE, the main driving loop which performs the syntax analysis. The recognition tables from the constructor are read-only data for this routine, which upon recognizing a syntactic construct calls:

(3) CODEGEN, the semantic routines.

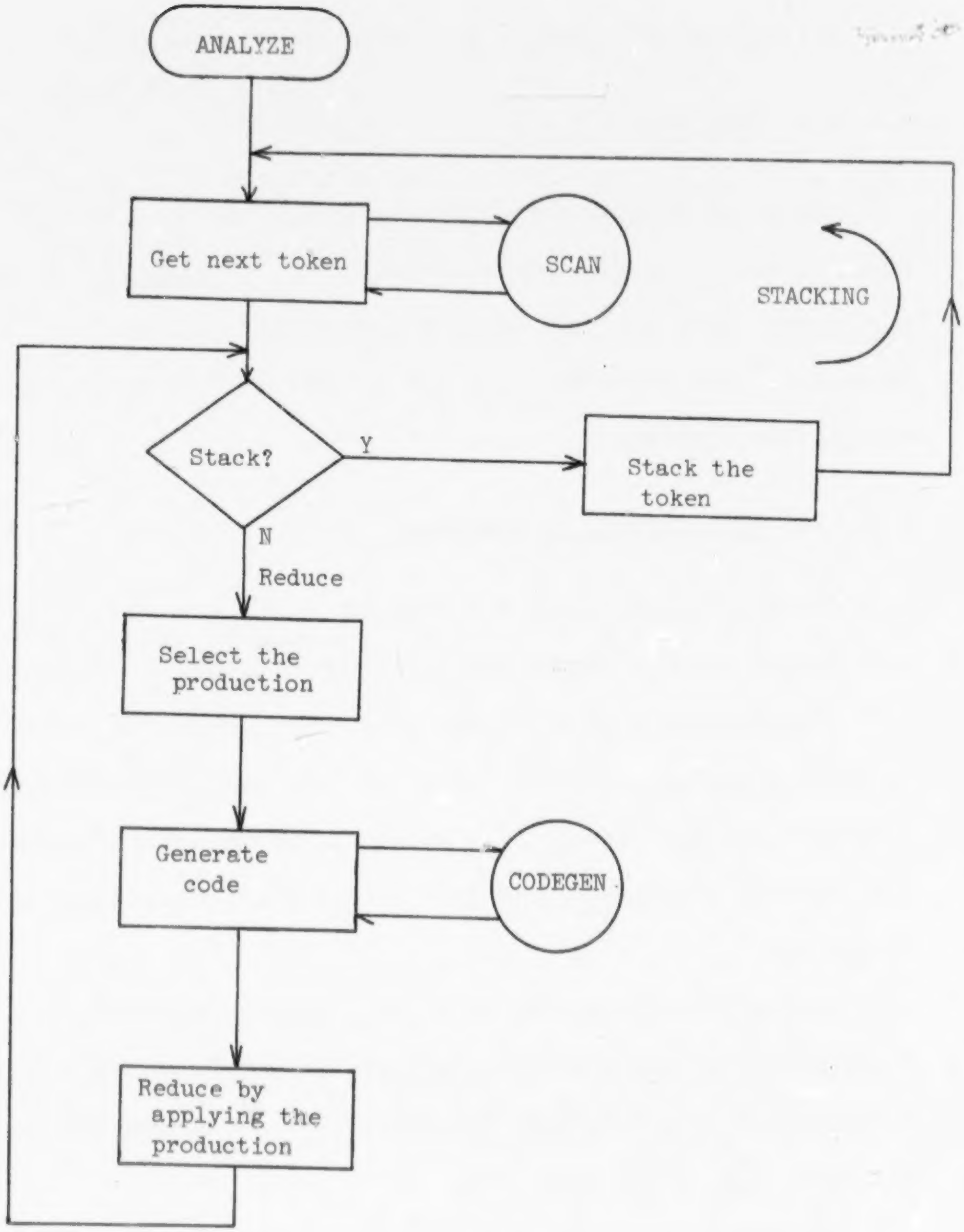

ANALYZE
Get next token
SCAN
STACKING
Stack?
Y
Stack the token
N
Reduce
Select the production
Generate code
CODEGEN
Reduce by applying the production

Figure 6.3 -- The main compilation loop in COMPILER showing the relationship between ANALYZE, SCAN and CODEGEN.

Their interrelationships are shown in Figure 6.3.

6.3.3.1 ANALYZE

This parsing algorithm is the nucleus of the TWS and is described in Chapters 4 and 9 of *A Compiler Generator*. The PL/I version used in COMPILER is included in Appendix D to this thesis. It is based on a PL/I version by Walters [1970].

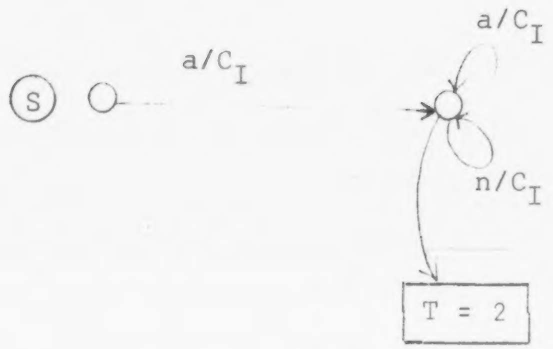
6.3.3.2 The lexical flowgraph

So that SCAN could be changed easily, a notation was developed for describing lexical analysis.

Cheatham's lexical graph [1967] has been modified so that advantage can be taken of certain properties of DIAL and so that there is a close correspondence between the lexical flowgraph and the actual PL/I code used in COMPILER.

A lexical flowgraph is a collection of nodes, directed line segments and labels; recognition of a token consists of a successful traversal of the flowgraph from the node Ⓢ to a node □.

Before the complete notation is introduced, consider a simple example. The recognition of the terminal symbol <identifier>, whose syntactic definition was given in Section 6.3.1, can be described by the following flowgraph.



Starting at Ⓢ, with register I set to null, an alphabetic character (class a) causes a move to the second node and Concatenates the character into register I. Then the graph loops on the second node, concatenating alphabetic characters and numeric characters (class n) into I, until another class of symbol appears. Then the traversal is complete at node □ where 2 is placed into the Type register to signify <identifier>.

There are three parts of the notation:

(1) Phrase structure grammar notation

V_N the set of non-terminal symbols.

V_T the set of terminal symbols, i.e., terminals as far as ANALYZE is concerned and hence sometimes referred to as tokens.

V_{T1} the set of direct terminals. A direct terminal is a terminal which is recognized directly by SCAN, e.g., GOTO, +, and <=.

V_{T2} the set of derived terminals. A derived terminal is one which is derived according to some rule operating on elements of the character set, e.g., <text constant>.

K a partition of the character set into classes:

a = {A,B,...,Z,a,b,...,z}
n = {0,1,...,9}
; = {;}
+ = {+} etc.

λ the empty string, of length zero.

(2) The communication registers in COMPILER

A set of registers is filled by SCAN for use by other parts of the compiler. The registers and their contents for the current implementation of DIAL are:

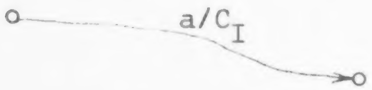
N BCD of number constant
S BCD of text constant
I BCD of <lexical identifier>
T type (1=element of V_{T1}, 2=<lexical identifier>, 3=constant)
D constant type (non-null only for T=3; 1=text, 2=number).

(3) Flowgraph nodes and labels

The start point Ⓢ has already been seen. At Ⓢ the registers N, S, and I contain λ.

LP pointer to the next character in the input string to be scanned.

labels labels on line segments are of the form k/A where $k \in K$ is the class to which the character belongs, and A is the action to be taken. A can either be the null action o, or C_R which means compose the character into register R, e.g.,



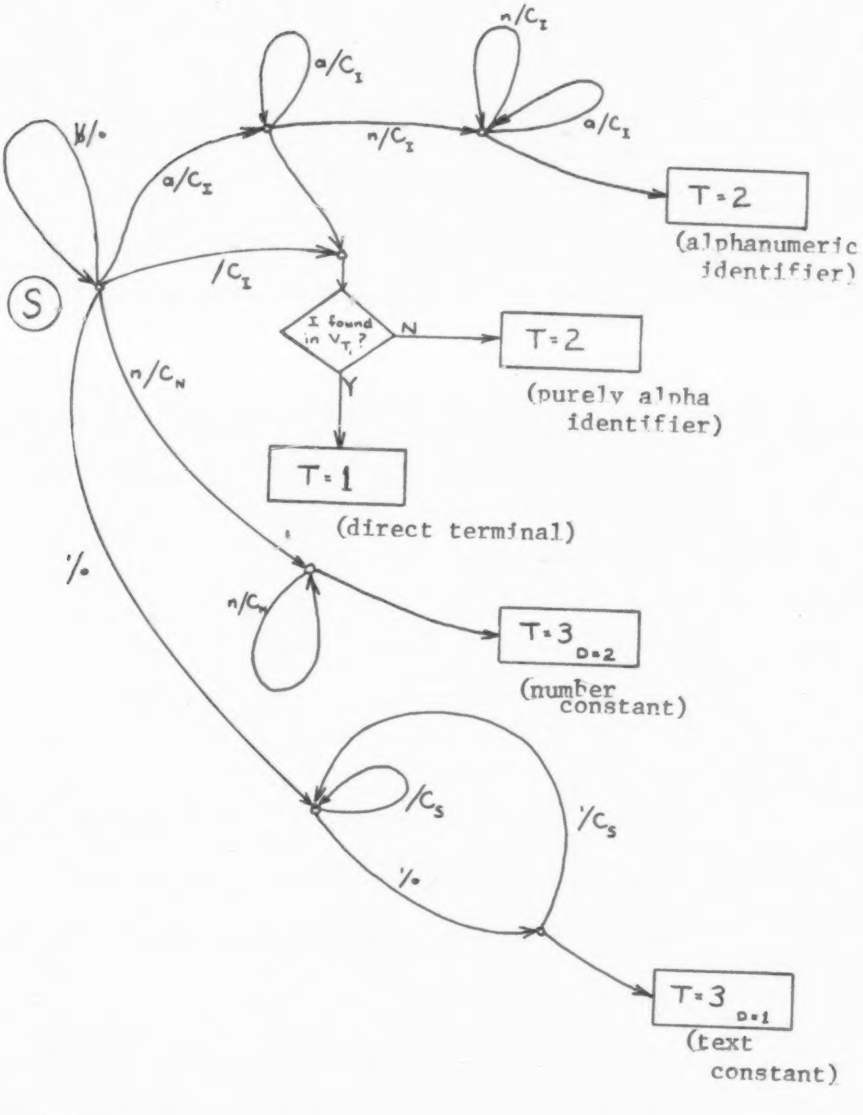
means "if the character is alphabetic then concatenate it into the I register."

An unlabelled line segment means "anything not accounted for."

▭ denotes end of traversal. An element of V_T has been recognized and the appropriate codes entered into the registers. The convention that LP is always set ready for the next traversal has been adopted.

◇ the normal program flowchart decision symbol.

Figure 6.4 shows the lexical flowgraph of the scanner used in an early version of DIAL.


n/C_I
a/C_I
b/·
a/C_I
n/C_I
a/C_I
T=2
(alphanumeric identifier)
/C_I
S
I found in V_T1 ?
N
T=2
(purely alpha identifier)
Y
n/C_N
T=1
(direct terminal)
'/·
n/C_N
T=3 D=2
(number constant)
/C_S
'/C_S
'/·
T=3 D=1
(text constant)

Note: Certain elements of V_{T1} ,e.g., <- and ¬=, are not recognized by this simple version. The current version recognizes them by extra line segments on the $/C_I$ path leaving Ⓢ .

Figure 6.4 -- A lexical flowgraph for an early version of DIAL.

6.3.3.3 CODEGEN

Recall from Figure 6.3 that CODEGEN is called by the main compilation loop just before each reduction is done. CODEGEN is highly systematic and modular: to each production in the grammar there corresponds one code section. Figure 6.5 shows code sections for three productions in a recent version of DIAL. Many of these code sections remain completely unchanged as DIAL is extended.

6.4 Steps in using the CAI translator writing system

A systems programmer using the TWS has available to him the program listings and other documentation in the Systems Programmer Manual [Mudge, 1972]. The following is just a list of the steps he will follow in modifying and extending DIAL.

(1) Express the grammer in BNF.[6]

(2) Debug the grammar using the XPL constructor.

(3) Make a final run with the PL/I CONSTRUCTOR to punch recognition tables.

---

[6] Inexpressables are handled in the usual *ad hoc* way by code in COMPILER (mainly in CODEGEN).

```
 /*****************************************************************
 P#(39):                                                          ****/
 /*      39 <MATCH PRIMARY>  ::=   <CHAR CONST>
                                                                   */
     CALL EMIT(READT,0);
     CALL EMIT(COMPARELIT,TOS_INFO2);

 DO_REF:   /* INSERT THE JE INSTRUCTION IN THE CHAIN AND EMIT IT      */
 /*** SINCE WE WON'T KNOW THE JUMP ADDRESS FOR JE UNTIL WE REACH THE **/
 /*** LABEL AT THE END OF THE STATEMENT, WE HAVE A REF_CHAIN LINKING***/
 /*** ALL THESE JE INSTRUCTIONS. THUS DO_REF HERE INSERTS THE JE IN ***/
 /*** THE CHAIN.
     I = REF_PTR;                                                  ***/
     REF_PTR = P_COUNTER + FREE_INSTN;
     IF M_FIRST THEN DO; /*FIRST <OPERAND> SO SIGNIFY END OF CHAIN    */
                    I = 0;
                    M_FIRST = '0'B;
                   END;
     CALL EMIT(JE,I);
     RETURN;


 /*****************************************************************
 P#(40):                                                          ****/
 /*      40 <GOTO ST>  ::=   <GOTO> <IDENTIFIER>
                                                                   */
 /*** NOTE THAT THIS CODE IS ALSO JUMPED TO FROM <LABEL LIST> PRODNS.*/

 IF TOS_TYPE = 9 THEN DO; /*PREVIOUSLY DEFINED LABEL SO EMIT THE      */
                           /*COMPLETE INSTRUCTION.                    */
                          CALL EMIT(JUMP,TOS_ADDR);
                          RETURN;
                         END;
 IF TOS_TYPE = 0 THEN DO;  /*IT IS THE FIRST REFERENCE TO A FORWARD   */
                           /*LABEL SO INITIALIZE THE FIXUP CHAIN.     */
                         CALL EMIT(JUMP,0); /*ZERO = END OF CHAIN     */
                         ADDR(TOS_INFO2)= P_COUNTER  + (FREE_INSTN-1);
                            /* POINT TO THIS END ELEMENT OF THE CHAIN. */

                            /* NOTE THAT P_COUNTER REFERENCES THE 0 OP-*/
                            /* CODE INSTRUCTION PREFIXED TO TEMP_INSTNS*/
                            /* BY AUTHOR.  FOR GOTO STATEMENTS,        */
                            /* (FREE_INSTN-1) WILL ALWAYS BE 1.  FOR   */
                            /* PRODUCTIONS, E.G., <LABEL LIST>, WHICH  */
                            /* JUMP HERE, (FREE_INSTN-1) WILL ONLY BE  */
                            /* 1 FOR THE FIRST LABEL IN THE LABEL LIST.*/

                         TYPE(TOS_INFO2)= 10; /*INDICATE IT IS A LABEL */
                                              /*NOW NEEDING FIXUP.     */
                         RETURN;
                        END;
 IF TOS_TYPE = 10 THEN DO; /*IT IS ANOTHER FORWARD REFERENCE           */
                         CALL EMIT(JUMP,TOS_ADDR); /*ADD EL'T TO CHAIN */
                         ADDR(TOS_INFO2)= P_COUNTER + (FREE_INSTN-1);
                            /*POINT TO THIS LAST ELEMENT ADDED.        */
                            /*** SEE NOTE ABOVE FOR TOS_TYPE = 0. *****/
                         RETURN;
                        END;
 ERROR = 26;    /*CONFLICTING ATTRIBUTE -- IDENTIFIER NOT A LABEL.     */
 RETURN;


 /*****************************************************************
 P#(41):                                                          ****/
 /*      41 <ASSIGN ST>  ::=   <VAR> <ARROW> <LOGICEX>
                                                                   */
 /** CHECK TYPE COMPATIBILITY
                                                                   */
     J=C_SYM_DOPE.TYPE(STACK.INFO2(SP-2));
     IF J=1 THEN IF TOS_INFO¬=T_TEXT THEN ERROR=E13;
                                         ELSE;
          ELSE IF J=4 THEN IF TOS_INFO¬=T_INTEGER THEN ERROR=E14;
                                                   ELSE;
                    ELSE IF J=6 THEN IF TOS_INFO¬=T_SLIDE
                                     THEN ERROR=E15;
                                     ELSE;
                                ELSE ERROR=E27;
    IF ERROR¬=0 THEN RETURN;

    J = C_SYM_DOPE.ADDR(STACK.INFO2(SP-2));



    IF TOS_INFO = T_TEXT THEN CALL EMIT(STORECH,J) ;
    IF TOS_INFO = T_SLIDE THEN CALL EMIT(STXS,J) ;
    IF TOS_INFO = T_INTEGER THEN CALL EMIT(STX,J) ;
    RETURN;
```

Figure 6.5 -- The CODEGEN sections corresponding to three productions in a recent version of DIAL

(4) If necessary, modify SCAN (by hand) in accord with the conventions of the lexical flowgraph.[7]

(5) Put the new recognition tables in PARSER[8] and run it with some DIAL source.

(6) Go back to step (2) if PARSER runs reveal a problem.

(7) Write the new parts of CODEGEN and debug piece by piece.

(8) If necessary, write additions to EXECUTOR's delta code interpreter and debug piece by piece.

(9) Perform final test.

6.5 The class of grammars acceptable to the translator writing system

A grammar is acceptable to the TWS if it is acceptable to the syntax analysis algorithm in COMPILER. The algorithm used is a Mixed Strategy Parsing (MSP) algorithm of degree (2,1;1,1) according to McKeeman's definition. The tasks of describing in detail the parsing algorithm and of giving an adequate explanation of the degree are beyond the scope of this thesis. Chapter 4 of

---

[7] In most cases, the only changes will be to the symbol table routine in SCAN.

[8] This is a routine in the CAI TWS which performs lexical and syntax analysis only. It prints a parse trace and is useful in planning CODEGEN.

A Compiler Generator gives several parsing algorithms and the rationale for the MSP approach.

It suffices to say here that the class of acceptable grammars is large because grammars which are bounded context of degree (2,1) are allowed, but the algorithm does not suffer from the inefficiency usually associated with extended precedence grammars. The improvement results from the mixed strategy approach: it uses a degree (1,1) stacking decision function with three values ("stack", "don't stack" and "conflict") and reverts to a (2,1) predicate only for pairs where the (1,1) predicate is undefined. The central idea of the MSP algorithm, then, is to use simple (small) tables to make as many decisions as possible but to extend the class of acceptable grammars by using more complex tables for the exceptional cases.

In practice, a user wishing to implement his own language will use the constructor and Chapter 7 ("Programming in BNF") of A Compiler Generator as aids to obtaining an acceptable grammer.

CHAPTER 7

EXPERIMENTAL METHOD AND RESULTS

7.1 Introduction

This chapter evaluates the CAI System in use in the actual environment for which it was designed.

Some systems are easier to evaluate than others: with compilers,time and space are measurable; with information retrieval systems, precision and recall are good measures. However, programming languages and man-machine environments are difficult to evaluate. Validating a system whose single consistent design aim was ease of use is especially difficult. This is because of the large number and diversity of human factors involved and the lack of objective measures.

The approach taken was to make systematic observations, both quantitative and qualitative, of student and author use of the system.

In the latter part of Summer, 1972, F. P. Brooks, Jr., Chairman of the Department of Computer Science, as course author, programmed course material for the subject matter taught in the first four weeks of COMP 18 and 19. These two course numbers are the humanities and social science sections of the beginning programming course taken for credit at the University. In the Fall one complete class section of 22 students took the CAI course. The class was the test group in a controlled experiment to compare learning performance on the CAI System with conventional classroom instruction.

The Fall experience was so encouraging that in the following semester, Spring 1973, the CAI System was used for production teaching of three classes totalling 79 students.

The combined Fall and Spring use represents 406 student hours of online production use of the system. This figure does not include those sessions of students who dropped or students from courses other than COMP 18 and 19.

7.2 Collection of student use data

7.2.1 Experimental design

7.2.1.1 The objective of the experiment was to determine whether, for the particular subject matter, students receiving instruction via the CAI System achieved equal or better learning performance than those receiving conventional classroom instruction.

If the learning performance could be shown to be at least as good, or better, then this would validate the design of the student-system interface. Furthermore, the Computer Science department would adopt the system for routine teaching of several sections of COMP 18 and 19.

7.2.1.2 The score on a posttest, a 50-minute, in-class, closed-book examination on the subject matter, was the measure to be used in formulating statistical hypotheses. Appendix C contains the examination.

7.2.1.3 The methods of instruction, two similar versions of conventional teaching and one of CAI, were assigned to complete classes as follows.

Methods	Control		CAI
	Instructor 1	Instructor 2	
Experimental Units	Class 1 (COMP 18-3, 218X-1)	Class 2 (COMP 18-1)	Class 3 (COMP 18-2, 218X-2, 19, 219X)

The first two classes were control groups; the third was the test group.

The conventional method

The usual procedure for teaching the course was followed: the instructors were graduate students in the Computer Science department; they conducted three 50-minute lecture-periods each week; and designed and graded their own assignments.

The CAI method

The students received all formal instruction via the CAI System in the CAI Center. A weekly, optional-attendance question period was provided by the instructor in charge of the class. Assignments were designed jointly by the course author and instructor and graded by the instructor.

At the first class meeting the course author was introduced to the students.[1] He addressed the class on the expected advantages, e.g., individualization, of CAI and discussed its answer analysis limitations. A good rapport was observed.

Work station resources were allocated as follows. Three concurrent hour slots from 9 a.m. to 5 p.m. were available for student use, except for the first week when two stations were available. As back-up there was one complete work station and two other complete CC-30 terminals. Stations were scheduled on a student signup board. Each student was allowed to sign up for no more than two sessions per week.

---

[1] I view this student-author link as an important human factor in a potentially depersonalized form of instruction.

One proctor, drawn from a roster of three, was on duty at all times. For experimental control reasons the proctor was not allowed to answer subject matter questions when called by a student. His function was to administrate and to assist the students when system hardware or software problems arose. A proctor log of problems was kept. At the end of each session a student received, from the proctor, copies of the slides shown in that session.

The CAI course had the following lessons.

Lessons	Subjects
a1 a2 a3 a4	The basic objects (integers and character strings), creation of variables (declarations), and assigning values to them (assignment).
b1 b2	The basic operations on integers and character strings.
c	A complete program and basic input/output.
d	Looping by DO-loop.
e	IF-THEN-ELSE for decisions and branching.
f	Documentation, nested DO-loops, and nested IF-THEN-ELSE.

The subdivisions within lessons a and b were necessitated by a software bug which constrained the maximum size of a lesson. This was corrected by the time lesson c was written.

7.2.1.4 Most experiments include undesired sources of variation which may affect the value of the variable being measured. In this experiment the identifiable extraneous variables were controlled as follows.

(1) Teaching ability of instructors

This variable was controlled by using different instructors for the two control groups.

(2) Subject matter coverage

The two control group instructors and the CAI course author agreed on a common syllabus.

(3) Posttest

The questions on the examination came from four sources in approximately equal proportion - the two control group instructors, the course author, and the instructor in charge of the CAI class. The originator of a question graded that question for all students. Students' answer sheets were identified by a number only, so that the graders could not tell to which class they belonged.

(4) Hawthorne effect

The students in all three classes were told that they were involved in an experiment to compare different teaching methods.

(5) "New gadget" effect

This variable was ignored.

(6) Inherent variability between students

Students vary, of course, in intelligence, attitude, year in school, aptitude, and other traits which may affect learning performance. At course registration time, that is, when students were choosing courses for the coming semester, they knew neither that an experiment was to be conducted nor that a new method of instruction was to be used. The assumption was made therefore that students were assigned randomly to the three classes.

(7) Prior knowledge of the subject matter

Based on a cursory study of students in previous classes, this was assumed to be negligible for all students.

(8) Scientific bias of investigator

I disqualified myself from proctor duty, grading the examinations, and other pedagogical contact with the students.

7.2.1.5 Sample Size

The sample size of approximately 20 seemed to be adequate to justify the assumptions of normality to be used in the analysis of the data.

7.2.2 The questionnaires

The questionnaires were intended to provide data to substantiate the assumptions made about student characteristics, for qualitative evaluation of the design of the CAI System, for improving the author's course material, and for improving the organizational aspects of production teaching.

Appendices A and B contain the two questionnaires. Questionnaire A was completed by all three classes at the class meeting following the examination.

Questionnaire B was for the CAI group only. Sections I, II, and III were completed during the same class period as Questionnaire A; Section IV was returned on the following class meeting.

7.2.3 Production teaching in Spring

Because the Fall experiment showed that the CAI System met the learning performance criterion, as discussed in Section 7.3.2, the system was used for three classes in the Spring semester. The objective of this next use of the system was to get more real experience rather than to conduct a controlled experiment. Because the emphasis was on teaching as well as possible, rather than on careful control for evaluation purposes, new freedoms to improve instruction were possible. For example, the proctors were allowed to answer subject-matter questions when called during a session.

Specifically, this additional experience with the system was intended to gather more questionnaire data, to enlarge the set of unanticipated answers for later course material improvement, to evaluate the effect of improvements suggested by the Fall experience, and to subject the system to a heavier workload, namely, three times that of the Fall.

Work station resources were allocated as follows. Four concurrent slots from 9 a.m. to 5 p.m. were available for student use. As back-up there was one complete work station and two other CC-30 terminals complete except for slide projectors. Seventy nine students were registered on the system.

There was one proctor, drawn from a roster of three, on duty at all times. During the first week, in anticipation of a high number of proctor calls, an additional Fall-experienced proctor was on duty; this extra help turned out to be unnecessary.

The same quiz and questionnaires were used. Several questions which were applicable only to the Fall, e.g., III.5 on Questionnaire B, were deleted. The arrangements for completing the questionnaires, taking the exam and grading the exam were the same as for the Fall except that I graded the course author's question (#4) on the exam. Because I used the same written grading standard as he used in the Fall, I feel that no scientific bias was introduced.

The following changes to the Fall system were made.

(1) Course organization and CAI Center operation

A 15 minute hands-on introductory session was given to each student on the first day of classes.

The role of the text book was clarified.

Work station assignments were made on the student signup board by the proctor ahead of the session-change hour so that the students themselves could take over a work station. This was done to reduce the amount of activity on the hour.

The constraint on the maximum number of sessions per week was relaxed from two to three.

(2) Course material

Only minor changes were made. They included coalescing lessons a1 through a4 into lesson a, and b1 and b2 into b. "Press INT to continue when ready" was added to those course messages which had been removed too quickly. The very few errors in the slides were corrected.

(3) The on-line system and work station

No design changes were made to the CAI System or to the work station.

The following implementation changes were made: CAIFILES was enlarged, the routine to handle the free block list was changed, and some of the known bugs were corrected. The terminal manufacturer improved the CC-304 light pen and corrected a design fault in the slide projector interface.

(4) The off-line system

Extensive additions were made to improve the daily reporting on student activity.

7.3 Analysis of student use data

7.3.1 Introduction

Results from both Fall and Spring are presented. However, only the Fall data are used in comparing CAI versus the standard method of instruction.

7.3.2 Fall posttest scores

The posttest scores have a maximum of 50. The means and standard deviations were

Conventional		CAI
Instructor 1	Instructor 2	
m = 34.2 s = 10.3	m = 31.4 s = 10.0	m = 34.5 s = 10.0
n = 16	n = 22	n = 21

Because of the large standard deviations, it can be seen, by inspection, that the null hypothesis that there is no difference in learning performance cannot be rejected. Homogeneity of variance is also obvious by inspection.

There is insufficient evidence to conclude that the CAI method is significantly better or significantly worse than the conventional method.

If indeed the CAI method is different, I have been unable to detect the difference because of the high standard deviations in the scores. There is no reason to believe that the standard deviation would be different in future experiments of the same experimental design. Thus, in order to detect a significant difference, if such existed, one would need a much larger sample size. For example, using a one-tail Student's t-test, for a difference in sample means of 0.5, a standard deviation of 10.0, and testing at the 95% level, n would have to be at least 2165 [Kirk, 1968].

7.3.3 Posttest scores for both Fall and Spring

For completeness the Spring scores are included to give the following:

Conventional		CAI	
Instructor 1	Instructor 2	Fall	Spring
m = 34.2 s = 10.3 n = 16	m = 31.4 s = 10.0 n = 22	m = 34.5 s = 10.3 n = 21	m = 35.0 s = 9.5 n = 64

7.3.4 Time data

The times for the CAI groups are from summaries of the daily log data. The control group time is computed as 10 lectures of 50 minutes each (I am assuming that each student who missed a class period spent the equivalent amount of his own time to catch up). The first class meeting is not included for any group. The times in hours and minutes are:

Control		CAI	
		Fall	Spring
constant 8:20	mean	4:34	4:01
	median	4:19	3:45
	range	2:10-8:43	2:08-8:56

The CAI times do not include the weekly question periods because it is difficult to assess the length and usefulness of these periods. In the Fall the periods terminated when student questions stopped; the three periods lasted approximately 20, 30 and 30 minutes. Moreover, the responses to question I.14 of Questionnaire B cast doubt on the usefulness of the periods. The freedom from experimental control in the Spring allowed the instructors to raise questions for discussion. They discussed problem areas exposed by the proctor log and the daily log from the system. In addition they lectured on algorithm design and the mechanics of running programs and interpreting program listings. Each of the three periods was the planned 50 minutes long. Using these figures as bounds we have

Control		CAI	
		Fall	Spring
constant 8:20	mean	5:54	6:31
	median	5:39	6:15
	range	3:30-10:03	4:38-11:26

I conclude, even with these conservative bounds, that the CAI method saved most students a considerable amount of time.[2]

A consequence of this time data is that the first part (CAI) of the next COMP 18 and 19 course will be collapsed from four weeks to three. In the Spring, two of the students completed the CAI course on the first day of the second week and 40 had completed it within two and a half weeks. The daily student report to the instructors shows which students need to schedule extra sessions.

7.3.5 Questionnaire data

Appendices A and B give the questionnaires and summaries of the student responses. A large portion of the data gathered pertains to quality of course material, reaction to the CAI method *per se*, etc.; hence their analysis is beyond the scope of this thesis.

---

[2] Similar time savings have been observed at other CAI installations.

The following conclusions pertain to the CAI System itself, as distinct from the course material. For most students, the system was transparent in the communication between author and student. The students felt they were concentrating on the course material, not the system (QB.II.17,[3] 22, and 30). The system was reliable (QB.III.4) and the response time was adequate most of the time (QB.I.17). The work station was firmly approved (QB.IV.5 and QB.III.16). However, QB.III.13 suggests that winter use requires better room ventilation. The mode and median times for students to get used to the operation of the system were both 10 minutes (QB.III.2). Assuming three contact hours per week, the majority of the students chose 2:1 as the best balance between CAI and in-class instruction (QB.I.15)[4]. Other questionnaire data are used in Chapter 8 to substantiate the discussion of design decisions.

With regard to the students' reactions to the CAI method as they experienced it in this setting, I conclude the following. The students enjoyed the scheduling freedom (QB.I.3), felt they could work at their own pace (QB.II.18), and felt that it made efficient use of their time (QB.II.25 and QB.II.31). They disagreed with the statements that (1) CAI is inflexible (QB.II.35), (2) CAI makes learning too mechanical (QB.II.19 and QB.II.40), and (3) CAI made them tense (QB.II.23 and QB.II.32).

---

[3] This is an abbreviation for Questionnaire B, Question 17 of Section II.

[4] This is an improvement over the Phase I system (Chapter 1) where a 1:2 ratio was chosen by the majority of its students.

The assumptions made about differences in student characteristics were substantiated. Questionnaire A shows no substantial differences in attitude or previous computer experience.

7.4 Author use data

DIAL and its operational environment were first put to production use in the preparation of the course material for the Fall experiment. The course author, Brooks, used the following approach for each session. Before he started a work-station session he would prepare final draft copies of each slide in the lesson. This defined the concepts and their order of presentation. With each slide, he prepared the questions to be asked, but not the answer analysis. Then, on the system, he keyed in the actual questions, and composed in DIAL the answer classification, feedback, and program sequencing. All composing of new DIAL code and debugging was done on-line.

Several problems, which were a consequence of this being the first use of the system, should be considered in interpreting the time data given below. The problems were:

1. A severe deadline

Lesson a was started on August 22 and had to be ready for students on September 4. The course was completed on September 20. This had the following consequences: (1) the scheduled pre-Fall-course testing of the course material was not done; (2) some Fall students missed sessions because the next batch of material was not ready; (3) the author had long, often five-hour work-station sessions.[5]

2. Software bugs

Software bugs are always expected in a new system; several minor ones were detected and corrected quickly. However, a bug in SOURCE, the routine which handles editing of an author's DIAL statements, sometimes scrambled the current copy of a lesson. The bug was elusive and the deadline intensified the anguish it provoked.

---

[5] Brooks's preferred session is 2 hours.

3. Recompile time

Because recompile time was long the author avoided frequent recompiles and hence lost some flexibility. This implementation, not design, problem is discussed in Section 8.3.

The course consisted of 66 pages (50 lines per page) of DIAL program listings and 64 slides. The mean student time for the Fall course was 4 hours 34 minutes. The total amount of author time, about 75% of which was spent on the CAI System, was 108 hours 30 minutes. Thus the number of author hours to produce one (work-station time) hour of course material was 24. To compare this with published ratios from other projects we must include the (estimated) time spent by author-support personnel. Typing the slide copy took 33 hours; photographic laboratory time was 8 hours; and delivery of)print listings, etc., took another 10 hours. Thus the ratio is 159:30/4:34 or about 35 author-hours per student-hour. This is a factor-of-five improvement over other reported results from systems of authoring (Section 8.3).

CHAPTER 8

DISCUSSION

8.1 Introduction

This thesis is concerned with designing and building tools - tools to

(1) build a CAI course for the beginning computer programming course at the University;

(2) serve as a framework for CAI research;

(3) study human factors in interactive CRT-based systems.

The single consistent design aim was ease of use - this implies that the system should be well engineered as regards human factors. By minimum impedance to students' learning, the system would be attractive for CAI research; by being easy to use it would lower the cost (in author time) of preparing instructional programs. The quantitative data set forth in Chapter 7 will perhaps be less valuable to a designer of such a system than a set of qualitative observations, opinions, and satisfactions and regrets derived from experience.

Hence this chapter discusses some of the decisions weighed in designing the language and its operational environment and gives opinions on the results.

8.2 The student-system interface

8.2.1 This interface should be as transparent as possible: the desired communication is between a student and an author via the author's instructional program. Hence the CAI System aims to intrude upon this communication as little as possible. Sackman, in reporting a large-scale experiment at the U.S. Air Force Academy, warns against complicated interfaces [Sackman, 1970:180].

> The array of special calls, instructions, and online procedures, requiring direct interface between the user and the computer, can be overwhelming for many neophytes.

The following measures were taken in the CAI System.

Debugged work station

The work station design was debugged over a period of eighteen months, during which time three successive prototypes were built.

Simple commands for getting on and off the system

All that the student enters is his identification number - the CAI System uses it to locate the course the student is taking and to set up the session using his position in the course. Leaving the system is achieved by the command)OFF.

8.2.2 To minimize the effect of system breakdown (hardware or software) on a student's progress and attitudes, frequent recovery dumps are taken during each session. If the system does break down, these dumps enable the System to restart the student, when he next signs on, at a point just prior to the breakdown, without his having to take any special action.

8.2.3 At all times that students are using the system a proctor is in the CAI Center building. The students requested his help, considered him essential, and approved of the proctor call switch (Questionnaire B III. 6, 7, and 8).

8.2.4 The response time of interactive systems is critical. Sackman's study of the SAGE air defense network introduces an important new conversational principle [Sackman, 1967: 436]:

> Human performance in man-computer dialogue will vary with the similarity of the responding computer system to the real time exchange characteristic of human conversation in situations closely related to the operator task environment. As computer response-time and message pattern deviate increasingly from realtime parallelism

> with the appropriate conversational and problem-solving norm, so will user performance deteriorate with regard to the achievements of system goals, leading to increasing compensatory, erroneous and maladaptive behavior toward the computer.

In short, people are happiest and most productive when their interactions with computers follow the same patterns as their interactions with people. The environment being paralleled in CAI is the learning environment - a mixture of classroom and self-study.

In a later study concerned with user characteristics in time-sharing systems, Sackman says [1970:27]

> Users with tasks requiring relatively small computations become increasingly uncomfortable as computer response time to their requests extends beyond 10 seconds, and as irregularity and uncertainty of computer response time increases; users with problems requiring much computation tolerate longer intervals, up to as much as 10 minutes for the largest jobs.

For example, while an author will be sympathetic to a longer response time if he has just entered a command, e.g.,)RESEQ, which he knows will cause heavy file manipulation, the student will be always expecting a response time consistent with his classroom experience.

In human conversations, people usually give some response to an utterance within a few seconds, even if a reply has to await longer thought. If this is to be paralleled by an interactive computer system, the same time patterns should be the target. It is somewhat more demanding than Sackman's 10-second figure.

For replies which will take longer than a few seconds to produce, the CAI System design requires that an immediate acknowledgement be given. For example, the pause between requesting CAI from the CHAT Monitor Table of Contents (MTOC) and the display of the sign-on message is eight seconds. Times up to 30 seconds have been observed during heavy activity on the host computer system. The acknowledgement

PREPARATION FOR SIGN ON
IS NOW TAKING PLACE

is given. Another example is the running acknowledgement during the recompile process in author mode.

In a student session, except for sign-on and sign-off, the system is providing for student execution of an instructional program. The response time then is less than one second. When host system activity is high, response time varies between one and four seconds. According to the students (QBI.17) response time is adequate.

To reduce the four-second response time, modifications must be made, not to the CAI System implementation, but to task dispatching in the operating system on the host computer. The relevant parameters include (1) the frequency of CHAT's time slice, (2) the duration of each time-slice, and (3) the dispatching priority assigned to CHAT.

Those aspects of the CAI System implementation which contribute to the fast response time are: compilative rather than interpretive execution of DIAL programs, efficient disk access, no magnetic tape input/output, and adequate workspace in main memory for each active terminal. The Phase I system's use of ISAM datasets clearly showed the need for faster disk work. In the current implementation, course material is held on a file with PL/I Regional 1 organization and the basic BDAM access method. The student and author record files, however, are ISAM processed. Since they are used only at sign-on and sign-off, their slower response time is acceptable. The other contribution to the longer response time at sign-on comes from initiation of the CAI System.

The time it takes to display output once it begins is also important. My observations of the Phase I system, and typewriter-based CAI systems at other installations, have shown that students often lost interest while waiting for long (several hundred characters) textual messages to come out. Here, reading is being paralleled. Since a student can read faster than a typewriter can produce output, the waiting is a system intrusion on his communication with the author. We eliminate this by our CRT terminal and fast communication lines.

8.2.5 A substitute for hard copy

Because of the transitory nature of the display on a CRT, a student does not have a record of his preceding work. However, on a typewriter terminal he does. He may use the hard-copy output for looking back during a session or for reference after the session.

During the exploratory stages in the design, I planned to develop special measures (initiated by student command) to substitute for the loss of hard copy. Later, however, R. O. Dearborn in an experiment at the University showed that such measures may be difficult to justify [Dearborn, 1970: abstract]:

> Three groups of students were exposed to the same computer-administered programmed instruction in numerical differentiation with different degrees of access to the output from the typewriter terminal. Analysis of co-variance showed no significant difference on posttest scores between students who were allowed to keep the output and those who were not, nor between those students who could look back during the session at previous output and those whose view was restricted to the most recent output.

I have not therefore provided any hard copy facilities for student mode. Authors, however, may obtain printed listings of their DIAL programs by the)PRINT command; they have routinely done so.

The students in the class trial indicated that they would have liked printed copies of the questions and answers (Questionnaire B, I.9).

8.3 The author-system interface

8.3.1 Author experience

8.3.1.1 One of the *a priori* design decisions (Chapter 3) was that authors would be the experienced master teachers themselves, without intermediary coders. The potential tutorial power of such a system is only realized when the master teacher himself sits at it and brings his experienced intuition into the detailed interplay with the conceptions and misconceptions of his students. This has far reaching consequences for system design.[1]

1. The system must be so simple that he *can* use it.

Because the professor is an occasional user of the system, he cannot justify a long retraining time at the beginning of each period of use. Design for any occasional user faces a much more stringent requirement for simplicity than that imposed by a full-time user, who can be expected to stay current in the details of even a complex operating procedure.

---

[1] The consequences apply equally to systems designed for other professional workers, for example, doctors and business executives.

2. The system must be so easy to use that he will use it.

Unlike the graduate assistant, the professor can choose whether he will use the system. If a system has many idiosyncrasies, is awkward, is slow, or requires him to be constantly referring to manuals, he will not use it. He will delegate the task instead. As a general criterion, a system such as this is well-designed when it quickly becomes transparent to the user, so his whole conscious thought focusses on the subject matter.

With regard to whether the author-system interface met its design goals, the following observations are significant.

8.3.1.2 The author work pattern has been described. Briefly, slides, text, and questions were prepared in advance. Less than one-fourth of the questions are multiple-choice; the rest require the student to construct a response. Nevertheless, the system power and ease of use was such that Brooks composed all answer analyses and all DIAL code on line; most of it about as fast as he could (touch) type. His learning time was short. The tool did not intrude - he concentrated on the course material and the course material alone while at a work station.

This tool transparency is reported not only by Brooks but by another faculty member who observed while Brooks worked. The instant replay via)xeq meant that he could see his errors; the system ease of use allowed him to correct them himself immediately and iteratively. Finally, the author-hours/student-hours ratio was a factor of five better than has been reported for other systems; this is discussed next.

8.3.1.3 Bunderson [1970:51] says of CAI projects:

> . . . especially those involving considerable text and display authoring in connection with the production of tutorial sequences. These projects can require 200, 300, or more hours of work on the part of a team of authors, instructional designers, programmers, and media specialists to produce a sequence that would take an average student only one hour to complete.

To produce one student hour of course material, Brooks spent 24 hours; the total team time was 35 hours (Section 7.4). This factor-of-five improvement over Bunderson's ratio of 200 is attributable to the following.

1. Author effects

The author is a very experienced teacher and his pedagogical philosophy was formulated before he began to write DIAL lessons. He touch-types.

This author experience meant that much of the usual course-material iteration had in effect been done over the years before the DIAL experiment was started. The system design objective, however, was precisely to harness such experience by offering the necessary level of ease-of-use.

2. System effects

The interactive operational environment gave him immediate feedback. There are two levels of immediacy in the feedback: firstly, the DIAL compiler checks each statement as soon as it is entered; secondly, the)xeq facility provides interaction with the course material just written. The programming and pedagogical errors so discovered are corrected immediately by the powerful editing operations.

Good lesson library service is provided by the command language.

The user-oriented design of the entire disk system presents a one-level store to the DIAL programmer and hides all management of disk storage allocation.

The DIAL language is also an improvement over those existing languages which were studied; this is discussed in Section 8.3.3.

3. Organizational effects

An author saves much time by not having to explain his ideas to intermediary CAI language programmers or to correct their misconceptions or awkwardnesses.

Support personnel were provided for typing the final slide copy, photographic processing, and delivery of)print listings to the work station. Such support is helpful, not hazardous, because it does not intrude on the author-student interaction.

Each of the three effects contributed to faster instructional program writing and debugging and to achieving essentially the final version on the first iteration. Class testing and study of the student errors contained in the protocols turned up only minor corrections to the course material produced on the first iteration.

8.3.2 The interactive environment

8.3.2.1 Figure 8.1 shows typical sequences of author actions during a session to prepare and test a lesson.

In contrast to the systems in Chapter 2, the author in DIAL does not switch back and forth between creation and alteration modes; both are done in one mode. At all times (other than when he is executing a piece of course material) he has the one display format before him and he is free to cause any action. A right parenthesis starting an entry signals a command; a statement number signals a statement. The statement may be a new statement, an insertion, or a change; these are distinguished merely by the statement number.

The inherent properties of the CRT are exploited for text editing. These properties are (1) a transient image and (2) a two-dimensional format (hence pointing carries more information content). Any statement on the CRT may be changed, by establishing the window around it, if it is not already there. Figure 8.2 shows a change being made to statement 536. (Since 536 was not already on the CRT, the author displayed it together with surrounding statements to provide context). If, during the change, extra room is needed for the enlarged 536, a *THROW* is requested.

Author action	System response
SESSION	
`)SIGN ON 246246246`	`)SIGN ON _`
	`J.C. MUDGE SIGNED ON AT 930` `)_`
`)les d`	
`)100,4`	`)_`
`100 S cend, 'First, let''s review the last slide of the previous lesson.'`	`100 _`
	`104 _`
Compose and enter DIAL statements sequentially	*(If no error detected)* *Next statement number.*
	(If error) *Diagnostic and then the cursor is placed under that statement part which is in error*
`)xeq`	
	Execution of lesson d
Interactive debugging loop -change one or more statements and)xeq to	

Figure 8.1 -- A typical sequence of author sessions.

```
    view the result.  The
    editing, )list, and )print
    commands are also used in this
    loop.

  )off
                                        SIGNED OFF AT 1140

SESSION
                                        )SIGN ON _
  )SIGN ON 246246246
                                        J.C. MUDGE SIGNED ON AT 800
                                        )_
  )dir
                                        A     B     C     intro     strings
                                        demol TEST  d
                                        )_
  )loa d
                                        )_
  )num
                                        520 _
  520 d7: RESUME
                                        524 _

    Compose and debug as
    in previous session

  )off
                                        SIGNED OFF AT 910
```

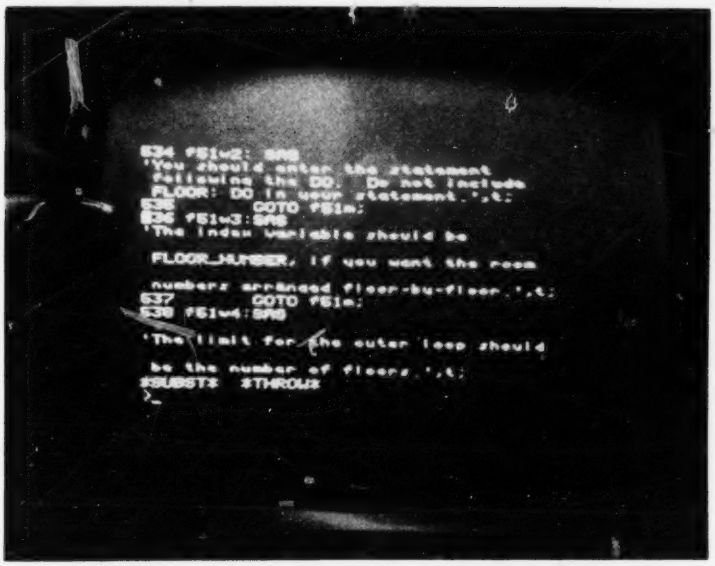
Figure 8.1 -- continued.

Continue with several sessions as in session above, until the lesson is ready for students

```
SESSION
    .
    .
    .
 )lid
                                  d
                                  )_
 )ren D
                                  )_
 )att d to PLC
                                  )_
 )dir
                                  A     B     C     intro    strings
                                  demol TEST  D
                                  )_
```

Figure 8.1 -- continued.

Author action	System response
)lis 534	
	Display screenful of statements, beginning with 534
Light-pen action as shown:	



```
534 f51w2: SAS
'You should enter the statement
 following the DO.  Do not include
 FLOOR: DO in your statement.',t;
535        GOTO f51m;
536 f51w3:SAS
'The index variable should be

 FLOOR_NUMBER, if you want the room

 numbers arranged floor-by-floor.',t;
537        GOTO f51m;
538 f51w4:SAS

'The limit for the outer loop should

 be the number of floors.',t;
#SUBST#  #THROW#
>_
```

Author action	System response
	Move window; place cursor as <u>5</u>36
Change statement 536,aided by cursor controls	
Advance cursor to end of window; press INT	
	)_

Figure 8.2 A change being made to a statement.

The *SUBST* function has not yet been implemented; even without it, editing proved to be very easy and smooth. Thus its implementation priority has been lowered.

8.3.2.2 Chapter 3 laid down certain goals for the interactive environment. How well are they met?

Source level

Inspection of the system commands shows that there are no commands concerned with the translation of an author's DIAL source to object code or with the manipulation, linking or loading of object code. All of his work is done at the source code level. He can therefore view the system as one which directly executes his DIAL statements. However, in explaining the response time variability over certain editing operations, one cannot avoid a discussion of translation. Thus, while an author can view conceptually the system as a DIAL machine if he so desires, the way he uses the system is influenced by the implementation; to date this phenomenon has been seen only with recompiling.

Anticipating an author's next move

After processing the contents of a window, the System tries to anticipate the type (statement or command) of the next action. It then repositions the window and the cursor within it to the most convenient place and generates part of the next action if possible. This both saves keystrokes and provides a time and action cue for the author. The following actions are taken. The author can of course override by keyboard action.

Case anticipated	Window, cursor, and cue generated by the System
command	)_
correction to statement (Cursor positioned under portion in error. For example, see Fig. 5.4.)	_
next statement in sequence	246 _
(can't predict)	_

Debugging an interactive system after it has been programmed requires not only the usual program debugging, but also a separate human-factors debugging that cannot be done on paper. Human-factors debugging requires the system designer to use his own system and to work with a few typical users. Brooks's use of the system showed that

CRT cursor positioning met my goal of anticipating an author's next move. However, it revealed a related bad awkwardness:)LIST as originally designed put author mode into a state which an author left by either exhausting the)LIST request or by entering)FINISH. Furthermore, if any other command was entered while in this state, the system responded with a fixed-time diagnostic forcing a wait of 5 seconds. It became clear that)FINISH and the special in-list state were both redundant and they have since been removed.

Minimizing user direction

Wherever possible, the design attempts to relieve the author from having to supply direction to the system. An author can be completely unaware of the existence of the five different logical files associated with each lesson - he views a lesson as a set of DIAL statements with a name. A directory in the File Maintenance System keeps track of a lesson's location, protects it from other authors, and protects it from tampering once it has been attached to a course. Neither need he be concerned with the system's use of backing storage - he views his DIAL machine as a one-level-store machine. Another example is that all source code entered is automatically saved against system breakdown -

he does not have to request such protection.

With the small number of functionally rich commands there is a price that the user must pay - he loses flexibility while gaining simplicity. For example,

(1) an author has no means of structuring his instructional programs and data so that they run more efficiently;

(2) there is no way an author can suppress the disk actions necessary for saving source code.

Such flexibility *vs* rigidity tradeoffs are made in any system design. The Job Control Language of OS/360 provides an extreme example. There is a large number of primitives and thus one can do almost anything; it is, however, difficult to use.

Experience-dependencies

The system is not responsive to changes in an author's skill in using the system. Whether he is new to the system or has been using it for several months he will receive the same level of diagnostic messages. Although this may annoy the experienced author, because of the speed of display of messages, he loses no appreciable terminal time.

An incremental compiler

The design called for an incremental compiler as the language processor for DIAL. The possible software implementations of a language cover a spectrum with an interpreter at one end and an incremental compiler at the other. The current implementation of DIAL is around the middle of these two limits in the spectrum - a fast batch compiler entered interactively. There are two major files associated with each lesson - the source code and object code files. When statements are entered sequentially, response time has a distribution skewed over 1 to 5 seconds with the mode at 2 seconds. This fast response time is due to overlapping disk work with user entry of the next statement, but, more importantly, new code is being added to the end of the existing object code file. However, when an out-of-sequence statement is entered, e.g., when an author is editing, such a change to the source code triggers a recompilation of the complete lesson and the building of a new object code file. Response time is then unacceptable - of the order of 20 seconds for a small (40 statement) lesson. However, when host computer system activity is high and a complete lesson must be recompiled, the author might wait 20 minutes. The current implementation does, however, provide a way of avoiding this long response time for each source code change. A *C* light button appears on the author mode display format.[2] Triggering of recompilation is suppressed by turning off *C*. Thus by batching his changes - entering them all with *C* off - he need only suffer the long response once, for his last change. This batching, in fact, matches how one thinks.

Nevertheless, the long recompile time is a severe problem for an author: it wastes his time, he loses flexibility during a session because he avoids frequent recompiles, and it discourages him from using the system during high host computer system activity. Improving the recompile time involves a major software change which could not be made during Brooks's use of the system because of the deadline. I did, however, provide a running acknowledgement (a "statement-number odometer") in lieu of a reply; this greatly improved the human factors.

An incremental compiler would avoid producing a new object code file for each source code change by structuring the object code file as a chained list, with each node being a set of object code instructions corresponding to one source code statement. This would provide the important fast response to author changes. It should be the next task undertaken in improving the implementation of the CAI System. Note, however, that such a chained structure can, by introducing another level of indirectness, result in a slower execution. So, at say)ATTACH time, all references should be resolved to absolute ones, and the code linearized, so that the execution speed is equivalent to the directly compiled code in the current implementation.

---

[2] It is not shown in the Chapter 5 examples.

Some reprogramming of the current implementation could result in a language processor closer in the spectrum to the incremental compiler. For example, the system could make some ad hoc determination of which parts of the object code file need not be discarded.

An interpretive implementation, while easier to build, was not used because of the execution-time cost in student mode.

Diagnostic messages

When an error is detected by the system, the ease with which an author can determine the true cause of the error is important. Two classes of diagnostic messages exist - those caused by errors in using the system commands and those caused by errors detected by the compiler. The messages produced by the compiler when it detects a semantic inconsistency have been carefully worded and are effective. For syntax errors, however, the characteristics of McKeeman's syntax analyzer are very evident - the diagnostic for the following error

```
MATCH x | L7
```

is

```
ILLEGAL SYMBOL PAIR: <M_OPLIST> _|_
```

I have not, in the current implementation, made any attempt to improve on such diagnostics. Improvements must be made, particularly on those diagnostics, e.g.,

```
NO PRODUCTION FOUND. IMPOSSIBLE
TO CONTINUE PARSE.
```

which do very little apart from signalling that an error has been detected. In practice, however, they are infrequent.

A general principle for systems on top of others is that diagnostics from the inner system must be translated into the terminology of the outer one before being fed to the user.

8.3.2.3 The following changes were made as a result of human-factors debugging.

1. The QAR screen division (Chapter 4) into automatically formatted question, answer, and response areas, was introduced into DIAL. The change was made over the weekend

after the first week of Fall student use. This was the most important human-factors change.

2. The)FINISH command was removed.

3. The identifier rule in DIAL was changed to allow lower case letters; similarly, the command language interpreter was changed to accept lower case as well as upper case.

4. The)LID command was added to help avoid erroneous (and disastrous) RENAME's.

5. The qualifiers to the)LIST and)XEQ commands were changed as follows

old design	current design
m thru ...	m
m	m.
m thru n	m,n

In practice, the first of these turned out to be most frequent by far, so its invocation was made most succinct.

8.3.3 The language DIAL

Here I will discuss the strengths of, the inherent weaknesses in, and the current omissions from the DIAL language itself.

8.3.3.1 Strengths

1. The language aims at ease of use by being syntactically consistent and by being procedure-oriented and problem-oriented rather than reflecting the underlying machine. This contrasts with many CAI languages, which are essentially assembly-language level.

The following comparative examples are used to illustrate this.

Subroutine linkage mechanism

The following Coursewriter example shows the linkage via return register 4 to the subroutine yesno

```
Invocation:            |           Comment:
                       |
  ld  next /r4         |      save return address
                       |
  br  yesno            |
                       |
next                   |
                       |
  qu                   |
   .                   |
   .                   |
   .                   |
Subroutine:            |
                       |
yesno                  |
  qu Type yes or no    |
   .                   |
   .                   |
   .                   |
   .                   |
  br r4                |      return to point of
                       |                invocation
```

The equivalent DIAL code is

Invocation:

```
CALL yesno
```

Subroutine:

```
yesno: PROC
   .
   .
   .
END yesno
```

Arithmetic

To perform the DIAL computation

```
E <- (A + B + C) * D
```

one would code the following in CW

```
                              comment:
ld c1/c5            |   load A (in counter 1)
                    |   into E (counter 5)
ad c2/c5            |   add B
ad c3/c5            |   add C
mp c4/c5            |   multiply by D
```

In TUTOR one would code

```
CALC F5 = F1 + F2 + F3
CALC F5 = F5 * F4
```

Screen formatting

Instead of supplying explicit screen formatting information (row, column coordinates) as in the CW II example in Section 2.2.3, the DIAL author merely formats the display the way he wishes it to appear to a student.

Thus the CW II example of Chapter 2:

```
dt 4,3//,3/Point to the name of the animal
dt 8,3//,3/that barks.
dt 14,10///□dog
dt 18,10///□cat
dt 22,10///□rat
```

in DIAL would be

```
SHOWAS

'  Point to the name of the animal
   that barks.

   *dog

   *cat

   *rat'
```

Branching decisions

Contrast the mnemonic value and naturalness of the DIAL statement

```
IF NWRONG >= 3 THEN GOTO Q5A
```

to the equivalent CW statement

```
br q5A//c3/ge/3,
```

or the equivalent TUTOR statement

```
JUMP I3,X,X,X,X,Q5A.
```

Generality in naming

In contrast to the fixed name assignments in CW, TUTOR, and WRITEACOURSE for counters, switches, etc., an author chosen identifier in DIAL can be used to name any data type whether it be a counter, register, switch, or buffer.

The naming of slides affords another example of this same point:

In DIAL

```
MVT=130;  /*Mean Value Thm diagram */
   .
   .
   .
SHOW MVT;
```

In FOIL

```
MVT=30
   .
   .
   .
TYPE *MVT
```

In CW

```
fp(p) 30
```

Mnemonics

(1) Machine registers or states

CW name	DIAL name
b0	ANSWER (ANS is accepted as equivalent)
p0	CASE
p1	SQZ

(2) CW restart points

If p13 is on, then the system will restart a student who has been signed off at the label defined in return register 5. Thus, a program in CW with restart points would appear as

```
              .
              .
              .
              ld arith1/r5
arith1
              .
              .
              .
              .
              ld arith2/r5
arith2
              .
              .
              .
              .
```

In DIAL this would be

```
          .
          .
          .
ARITH1: RESUME
          .
          .
          .
ARITH2: RESUME
          .
          .
          .
          .
          .
          .
```

String manipulation for presenting text

As pointed out in Section 2.4.2, existing languages have neglected the potential of a computer for presenting text. Consider, however, just the simple ability to name an often-used text, e.g., the statement of a theorem to be used in several distinct paths in a frame. In DIAL this would be done by

```
        THM  =   'The theorem states
    that ---------------------------
    --------------------------------
    --------------------------------
    ------------------------------'

      .
      .
      .
      .
  SHOW THM
      .
      .
      .
      .
  SHOW THM
      .
      .
      .
      .
  SHOW THM
```

With no such ability to name a character string, as is the case in almost all author languages, the author must type the complete theorem whenever he needs it in his program.

Note however that with CW the determined author can avoid the retyping by using buffer storage as a temporary naming facility.

For example

```
ld The theorem states that
   -----------------------
   -----------------------
   ----------------------- /b2
   .
   .
   .
   .
 qu b2
   .
   .
   .
   .
 qu b2
   .
   .
   .
   .
 qu b2
```

However there are two restrictions on this type of programming - there are only five buffers and each one can hold only 100 characters. Multiple constants can be stored in one buffer and then fetched by a substring operation. The program then is hardly straightforward.

Clean syntax

Chapter 3 argued against adapting another language, using the possibility of awkward syntax as one of the arguments. The HumRRO Project IMPACT author language [HumRRO, 1970] is an extension of CW III which has better text manipulation facilities. The awkwardness of the extension can be seen in this example [HumRRO, 1970:25]

```
qu ((DIS D260,1),(SET GLOS=0))
```

which is an IMPACT-Coursewriter instruction to retrieve a display from a text file prepared independently on a text editing system.

2. Subroutine facility

The subroutine concept is an important contribution of computer science to the design of algorithmic processes. It should be as useful in instructional programming as in conventional computer programming, once an author understands the invocation and parameterization mechanisms.

3. Text constants

The text manipulation facilities of DIAL can be used for preparing text arguments for SHOW-statements. For example, consider the following program segment.

```
OBS <- 'Study the slide above.'
PINT <- 'Press INT to continue when ready.'
PENIND <- 'INDICATE YOUR ANSWER WITH THE LIGHT PEN'
  .
  .
  .
  .
SHOW OBS, MVT  /*Show Mean Value Thm slide */
  .
  .
  .
SHOW OBS || PINT
  .
  .
  .
SHOW OBS, PENIND
  .
  .
  .
```

Note that the assignment statement is only being used in this example to name a string of text, not to assign a value to a variable. Such a text appears only as read-only data in the program. However, once a variable is set up in this reentrant program environment,[3] it must be kept in each activation record, i.e., there must be one copy per student.

This special nature of text data has been recognized and capitalized upon to improve the efficiency of the implementation.[4] By using a naming-statement (=) instead of assignment (<-) an author assures the compiler that the target symbol is constant, not a variable, and hence read-only. Such text (or slide) constants become part of the fixed part of the reentrant program representation. There is only one copy no matter how many students are active.

---

[3] I am referring to the reentrant program representation of authors' DIAL programs running on the DIAL machine, not the CAI System, which is also reentrant.

[4] In a multi-pass compiler (DIAL has a one-pass compiler) this could be done without user help.

The above example then becomes

```
OBS = 'Study the slide above.'
PINT = 'Press INT to continue when ready.'
PENIND = 'INDICATE YOUR ANSWER WITH THE LIGHT PEN'
    .
    .
    .
    .
SHOW OBS, MVT
    .
    .
    .
SHOW OBS||PINT
    .
    .
    .
SHOW OBS, PENIND
    .
    .
    .
```

4. <u>The generality of a programming language</u>

The features in DIAL that give the generality mentioned in Chapter 3 are the

(1) generality in resource allocation

(2) naming generality

(3) PAT system matching function

(4) CALL statement

(5) character string operations

(6) SHOWAS statement allowing arbitrary formatting

(7) one IF-THEN-ELSE for all kinds of tests

(8))INCLUDE facility for library material.

5. Consistency

Examples of the consistency argued for in Chapter 3 are

(1) modality, e.g., CASE and SQZ, are uniformly treated,

(2) naming of character string and slide constants,

(3) expressions, e.g., wherever text may appear, a text expression may appear.

Note that these measures set a precedent for consistent extensions to DIAL.

In trying to achieve consistency I found the formalism of a grammar for the language to be very helpful. As a guide to the ease of use of a particular syntactical construct, I found the number of productions to be useful.

8.3.3.2 Weaknesses

1. No timed response facility

There is no provision in DIAL for an author to get control back from the student if he has not answered a question within a certain time period. Although this is consistent with my main client's pedagogical philosophy of not wishing to pressure the student, it is a weakness in the language because I preclude the use of timed responses by other authors.

Another author use, which is pressure-independent, for a timed response facility, is in tailoring the course material to speed of student responses.

Note, however, that since the CHAT interface does provide tools for these facilities, DIAL could be so extended. An author does have the data in the log file as to how long students take to answer each question.

2. PAUSE

The rate at which succeeding units of instructional program output are given depends on whether or not a user response separates the units. When there is no intervening response, an author must be sure that PAUSE is set appropriately and so has to make the (sometimes difficult) judgment on how long the average student will take to read a unit. He may not, however, agree with this style of progressing a student, and instead, prefer that the student indicate when he is ready for the next unit. Perhaps a new statement is needed in DIAL, one which displays 'Press INT to continue when ready'. The same effect can be achieved in the current design, not by a special statement, but by a CALL to procedure INT, where INT is defined as:

```
INT: PROC
  REPEAT
   S'Press INT to continue when ready.'
  UNTIL PAT('')
END INT
```

Class use showed in fact that such student-initiated progression is better. The course material was so changed for the Spring use. A new statement, PINT, has been added to DIAL; this temporary language change will remain until procedures have been implemented.

3. Data types

The data types in DIAL are

text
slide
integer
label.

The data structures are scalar and vector.

These were regarded as essential for an author language. Other data types, for example

arrays
global text variables[5]
real numbers

---

[5] Note that global text constants are provided by the)INCLUDE facility.

are not in DIAL. The usual tradeoff between usefulness and cost of implementation (size of the main-memory-resident language translator) excluded them.

8.3.3.3 Omissions

The following are designed omissions.

1. A HINT facility

The HINT (or HELP) facility is usually designed so that a student may request a hint for each question presented. Hence an author must prepare a hint action for each question - if he does not, the student may be adversely affected by the standard system response 'NO HINT FOR THIS QUESTION'. I prefer amplificatory sequencing to be explicitly programmed. For example, in DIAL, use

```
UNREC *, L2, L3
```

with the code sections at L2 and L3 being amplification of the question. Furthermore, I believe that being able to request a hint or help at every interaction in a teacher-student dialogue is not natural.

Notice that HINT requires an author to define two teaching tactics for every micro-point. As contrasted with such preparation of both a coarse-grained teaching tactic and a fine-grained one, it is less costly for the author to put all students through the fine-grained one; and if display and response are fast, it may not be any more costly and tedious to the good student. (But then again, it may.) For the poor student, the success psychology is considerably better than failure psychology.

2. A calculation mode

Outside of the CAI System, our students have ready access to many terminals providing APL. Moreover, it is planned that some interactive PL/I service be a subsystem of CHAT. Perhaps the omission in DIAL of *calc*, is really a question of subject matter bias. *Calc* is important to PLANIT because it is directed mainly at statistics; the numeric tests for equivalence in PLCLS [Oldehoeft, 1969] provide an attractive form of answer analysis for numerical analysis. The UNC CAI Project intends to add a facility which is attractive for answer analysis in computer programming - a PL/I language processor (Chapter 9).

8.3.3.4 General comments

1. The DIAL subset used

The following are not in the current implementation of the DIAL language specs: vectors, the FRAME-statement, procedures, SUBSTR, INDEX, LENGTH, and IF-THEN-ELSE.

The language features actually used by Brooks were: naming, assignment, SHOWAS, SHOW, MATCH, PEN, PAT, UNREC, GOTO, RESUME, CASE, ENDLESSON.

2. Reserved words

Reserved words are those appearing in the grammar, e.g., SHOW IF MATCH . They cannot be used in any way except in their intended structural use in DIAL, a consequence of the implementation based on McKeeman's Translator Writing System. The effect of this limitation can be summarized by saying that it violates the criterion of modularity stated in [Radin and Rogaway, 1965]: ". . . but if you don't need it you don't have to specify it or even learn about its existence, ..." as one of the design criteria for the language PL/I.

This criterion, however, is not really applicable to DIAL. There are only 40 reserved words; an author can learn them all, a virtually impossible task for PL/I.

3. Two branching statements

Branching conditions and actions can be described by the IF-statement alone. However, because of the very frequent occurrence of

(a) a logical expression with the ANSWER register as a comparand

(b) responses which are equivalent for branching purposes

the MATCH-statement is provided to give a more compact notation, at the expense of mnemonic value.

The following two DIAL program segments produce the same result.

```
IF ANSWER = '2A' | ANSWER = '2C' THEN GOTO L1
IF ANSWER = '2B' THEN GOTO L2

MATCH '2A' | '2C', L1
MATCH '2B', L2
```

4. The)INCLUDE facility

The design called for a library subroutine facility whereby subroutine procedures are made available in some library for inclusion as procedures in an author's instructional program. This facility is widely accepted in computing and has been used in most computer installations for several years.

Rather than a library subroutine facility, the CAI System has the)INCLUDE command, a facility which is conceptually more powerful. It is more powerful in the sense that arbitrary text, not just subroutines, can be included.[6] Moreover, it is consistent with the DIAL machine concept - an author manipulates only source text, not compiled code as well.

---

[6] As an analogy, the PL/I compile-time %INCLUDE will accept arbitrary text, whereas the OS/360 linkage editor accepts only valid subroutine procedures.

8.3.3.5 Summary

DIAL is an improvement over existing author languages. The improvements can be seen both in the language itself and, more importantly, in the operational environment in which it is embedded - an integrated, functionally complete system serving authors, students, proctors, and computer programmers. Implementation with a translator writing system has been seen to simplify the remediation of weaknesses as they are discovered. As can be seen, DIAL offers little new in CAI function. However, some general, powerful, and easily used mechanisms have been borrowed from general programming languages, and embodied in a consistent syntax to provide features not usually seen in CAI languages.

8.4 The computer programmer - system interface

The interface discussed is the one involved in using the CAI Translator Writing System. Chapter 6 should help the reader in his assessment of the flexibility of the DIAL implementation.

Of the three parts of a compiler for a new version of DIAL, the lexical and syntactic parts are taken care of by the TWS. The semantic routines, however, require a PL/I programmer with knowledge of the internal structure of COMPILER. To assist him, the design of that routine is highly modular, particularly in the semantic routine CODEGEN.

The CAI TWS has proved useful in the following situations.

1. Progressive implementation of a fixed design

Several versions of DIAL, embodying progressively larger subsets of the specifications in Chapter 4, have been implemented.

2. Improvement in design

The language design process did not stagnate during progressive implementation, and several improvements have occurred. One replaced the CC and SC attribute declarations by the naming statement. For example

```
OBS = 'Observe the slide'
```

used to be achieved by

```
DCL OBS CC 'Observe the slide'
```

This change was handled entirely at the syntactic level by the CAI TWS. Another improvement, but one which requires some semantic change, was the generalization of mode switching. For example

CASE <- arithmetic-expression

replaced the two statements

CASEON and CASEOFF

The CAI TWS promises to prove useful in two other situations as well.

3. Correction of mistakes

Of those mistakes discovered during human-factors debugging, for example, the redundancies discussed in Section 9.2, DIAL/2, the majority of them are correctable at the syntactic level.

4. Extension beyond specifications in Chapter 4

This is, of course, the situation in which the TWS is most attractive. However, most of the computer programmer's work will be in writing and debugging at the semantic level. DIAL/2's PARSE function is an example.

DIAL/2 is a language radically different from current DIAL. But the differences are predominantly syntactic, not semantic. This further justification of a TWS implementation of DIAL was unexpected because I had predicted that DIAL would have semantic, not syntactic, limitations.

A good measure of the effectiveness of the computer programmer-system interface could be obtained when the next step in progressive implementation is taken. For at each step the changes are well defined. Time and effort data should be gathered for each of the following tasks, which are defined in Chapter 6:

language definition;

BNF programming;

PARSER runs;

writing CODEGEN additions;

writing new delta code interpretation in EXECUTOR;

debugging.

8.5 Observations about human factors

This section presents a selection of the more important human-factors considerations involved in my research. I believe, but have not proved, that these observations are generalizable to the design of most man-machine interfaces.

8.5.1 A human-factors debugging period

No matter how carefully a design is done on paper, such a debugging period is essential. It seems doubtful if modelling techniques for man-computer dialogues will ever eliminate the necessity for a prototype. Dialogues are highly dynamic and complex; paper design is inherently static.

Actual use of a system reveals awkwardnesses, inconsistencies, and redundancies (Section 8.3.2.3). Moreover, some of the assumptions on which the design was based, e.g., frequency-of-use and sequence-of-use assumptions, turn out to be wrong. On the other hand, solutions to problems (both the newly discovered ones and those recognized but not solved in the paper design) can be visualized more easily with a working system.

This observation has implications for system building: the schedule must contain a human-factors debugging period; and the implementation must be flexible. I would like to have had a TWS for the operational environment such as Newman's [1968], not just a TWS for the DIAL language.

8.5.2 Consistency (or uniformity)

To the extent that command languages, programming languages, and operating procedures deviate from the rule "the same things should be done in the same way wherever they appear", they violate the psychological principle of uniformity. They will usually be more difficult to learn and more difficult to use without error.

Several components of language are involved: syntactic, e.g., consistency in display format[7]; semantic; pragmatic, e.g., consistency with the user's real-world experience; stylistic, e.g., tone of messages; and lexical, e.g., rules for forming abbreviations.[8]

A consequence of the need for consistency is that instant design cannot be done; any change generally reverberates throughout the whole design.

Consistency throughout is part of the conceptual integrity of design. This integrity is far simpler to achieve when there is only one system architect specifying all the elements of the user-system interface. Two designers may agree completely on principles; there will still be differences in style, and these differences will inevitably show in the micro-decisions of the design.

---

[7] The student user of the CAI System has just one display format, QAR, to master, from sign-on through instruction to sign-off. The author has just two: QAR and the author-mode format.

[8] Sometimes, to maintain consistency, a designer will include facilities which he predicts will never be used. The m,n option on)list is an example.

8.5.3 Sackman's conversational principle

This principle (Section 8.2.4) is a very useful guideline.

However, response time, while critical, is not the only factor in paralleling human interactions. I suspect that because response time is readily measurable (compared to other factors, such as command language ease-of-use), it has been overemphasized in the literature.

8.5.4 Top-down design

Top-down design, or outside-in in this case, helps to ensure that the design focus is on the man in the man-machine interface, not the machine or implementer.

8.5.5 User-initiated progression

There are now very few intervals when messages are displayed for a fixed time in the CAI System; in most cases, and always for long messages, the user, not the system decides when to progress. This change in philosophy evolved during testing. It turned out that when a message is displayed for a fixed time, the slow user will miss part of it, and the fast user will get impatient. If an author's course material does this, it not only reduces individualization, but also frustrates both fast and slow students; the same applies to the system in which the course material is embedded.

The same is true in author mode, where the messages are generally diagnostic messages. An experienced author detects the cause of his error very quickly and wants to get on with fixing it.[9]

8.5.6 Minimality - the search for powerful primitives

As with computer architecture, programming language design, and command language design, the search is for powerful, general-purpose primitives. Sometimes these are found by seeking them to start with. Thus, a goal continually pursued in this research was a single canonical form for answer matching. The pursuit of this goal (not yet achieved) led to development, for example, of the PAT function, which subsumes the functions of MUST, CANT, DIDDL, keyl, etc., found in other languages. Often such primitives are arrived by iteration, e.g., collapsing several commands into one.

8.6 The cost of designing and implementing the system

Developing the CAI System used for the Fall 1972 class took 2358 man/hours over a period of three years.

---

[9] Fixed-time diagnostics annoy the experienced user in the same way that long typewriter-terminal diagnostics do.

This includes both the design of the system presented in this thesis and the implementation and documentation of the subset defined in the Systems Programmer Manual [Mudge, 1972]. It does not include the thesis writing time.

The size of the CAI System, i.e., the on-line routines, at the end of August, 1972 was as follows.

11,000 printed lines of PL/I source code
(including blank lines for readability)

3,691 PL/I statements

About 200 of the 3691 statements are DECLARE's, almost all of which declare many identifiers, which leaves about 3500 statements.

Hence, the 11,000 printed lines are made up of comments, blank lines, declarations, and about 3500 executable statements.

As with any software engineering project, the time includes the following activities.

1. Building scaffolding

   Apart from the usual scaffolding there was

   a. CC-30 i/o simulator (the CHAT interface simulator)
   b. PARSER to debug the syntax analyzer and to aid in using the TWS

c. offline service routines
   - crude versions of file maintenance of STUREC, AUTHREC, CAIFILES
   - lesson listing: an offline)PRINT command

2. Scrapped code
3. Project meetings
4. Demonstrations and discussions for site visitors
5. Systems Programmer Manual
6. Test programs for PL/I
   - learning language features new to me
   - testing features obscurely described in manuals
   - choosing among alternative PL/I methods

The time also includes the student/author workstation design and prototype construction.

Because of the method I use to record my time (effective hours), each time figure given in this section should be inflated by 20% to account for coffee breaks, etc., if they are to be compared to industrial work hours.

The high proportion of time spent in a project like this on design, tooling, and scaffolding is reflected by the fact that over one half of the 3500 instructions were written and debugged in the Fall semester of 1971. These 1800 instructions were done in 594 hours. This was straight coding and debugging after all design, scaffolding and data structures had been taken care of. For this sprint period, productivity was at a rate of about 5000 debugged PL/I statements/man-year. The overall project rate is about 2470 statements/man-year.

8.7 Is it widely applicable?

Could another university install the CAI System? Could a high school install a work station with access to DIAL at a remote host computer?

Given the properties of current time-sharing systems, the most portable system would be one which is written in FORTRAN and which uses a teletype as terminal. Indeed, the FOIL, WRITEACOURSE and PLANIT systems of Chapter 2 state this sort of high portability as a design requirement.

These constraints were rejected as too rigid and awkward for this project, but the same sorts of portability were goals.

8.7.1 Portability of the software

(1) The implementation language

The CAI System is written entirely in the machine independent language PL/I, and in this respect it is highly portable. The particular implementation used is the F-level OS/360 compiler. Changes may have to be made to run with other PL/I environments because

a. some implementations are a subset of PL/I (F).

b. the run-time environment of PL/I (F) is intertwined with OS/360.

c. the machine-dependent UNSPEC function of PL/I is used in the CAI System.

(2) The host computer's operating system

An absolute requirement for CHAT is OS/360 with the MVT option. Only minor changes need to be made locally. Without CHAT the prospective user is faced with two problems: communications support and multiterminal support. The communications support for single terminal operation would be reasonably inexpensive to build. This is certainly not the case for multiterminal support, the major part of CHAT.

8.7.2 The host computer

- A medium scale computer, e.g., a S/360 Model 50, would provide the amount of main memory and type of disk backing storage needed for the on-line routines of the CAI System. Thus the system is expensive, but a price must be paid for richness of function and good performance. Main memory usage is one copy of the reentrant load module (155,000 bytes) and a work space (the activation record) for each active user. The latter is dynamically allocated by the CAI System; space is requested when needed and freed when available. This results in a substantial improvement in memory use over static allocation. However, this dynamic behavior makes measurement difficult. The lower bound of each student work-space is 21,000 bytes; the upper bound throughout a session is about 25,000 bytes except at sign-on and sign-off times when 42,000 bytes[10] are used. The overall utilization of memory could be improved if work spaces were swapped out during user think times; the resulting response time degradation would not meet my design goals, however. The lower bound of an author work space is 10,000 bytes; the work space grows to 14,000 bytes when a statement is being compiled, and peaks at 26,000 bytes during author execution of a DIAL program.

Schultz's CHAT monitor has been carefully crafted in assembler language and requires only 28,000 bytes.

8.7.3 The terminal hardware

The student/author work station is built around a terminal which, if not in widespread use, is commercially available and based on established technology.

---

[10] 26,000 bytes of this are used by the OS/360 input/output routines, e.g., the indexed sequential access method routines. They are required only at sign-on and sign-off.

8.7.4 The communications hardware

To achieve a performance consistent with the design, remote terminals must be linked by a line of at least medium speed (2400 bits per second). Teletype-speed lines are certainly inadequate. Since our terminal and communications equipment from Computer Communications, Inc., can be configured flexibly (with the addition of a CC-70 communications processor if necessary), terminals in widely separated sites are possible.

CHAPTER 9

SUGGESTIONS FOR FUTURE WORK

9.1 Introduction

This chapter recommends several directions in which the present CAI System design might be extended, as well as presenting some research topics. Recommended improvements in the current implementation of the design are not discussed, as they are covered in the Systems Programmer Manual.

In contrast to the specific suggestions in this chapter, the goals of Phase III of the CAI Project provide more general direction. Phase II[1] produced Schultz's CHAT monitor, the CAI System, and Brooks's course material. As the Project formulates the goals for the next phase, we find that we are more interested in results that teach us something about interactive systems in general than we are in attempting to devise new CAI methods and strategies as such.[2] The West House facility will be used to investigate the factors in system design which make interactive CRT systems easy or hard to use. This includes work-station design, design of application-oriented languages, system command languages, system robustness and graceful-fail features, operational procedures, and integration of human assistants with machine systems. This thesis has investigated these factors severally, but their integration and interaction ultimately determines ease of use.

---

[1] See Chapter 1 for its goals.

The wide class performance variances observed in our tests show that a new methodology for evaluating such factors must be developed; probably it will be based on systematic observation of users, error-rate data, timing data, and user questionnaire data.

9.2 DIAL/2

In contrast to on-line entry of a program written prior to an on-line session, on-line composition places greater demands on a language. A much clearer understanding of these demands came from actual use of the system.

---

[2] Note, however, that in the pure CAI area, the Project has made available a proven, total, flexible system for teaching. Already several researchers in the University and in the state have shown interest in using the System for research in instructional programming, learning theory, and cost-effectiveness of CAI.

The proposals in this section reflect this experience and also the influence of Dijkstra's structured programming [Dijkstra, 1970].

The exact syntax and semantics of the proposals are not given. Moreover, their smooth integration with current DIAL to maintain a consistent uniform design is not covered at all.

9.2.1 Response sets

The response set provides a more powerful means of classifying responses in a MATCH-statement. The extra power comes both from the increase in function and from its generality. A set is defined by

set-name = <elements>.

As an example

MATCH set_1|set_2| <'Y', 'YES'>, next

would branch to next if at least one of the elements in the three sets matches.

Examples of sets are

1. set_1 = <'INTEGER', 'AN INTEGER'>
2. set_2 = <'CHARACTER STRING', 'A CHARACTER STRING'>
3. set_12 = <set_1, set_2>
4. numeric = <'0', '1', '2', ... '9'>
5. yes = <'Y', 'YES'>

```
6.  oddpen = <PEN(1), PEN(3), ...PEN(9)>
7.  gamma  = <'DOG', PAT('¢DOG¢'), P4>
```

Example 3 names a set which is the union of two other sets; any number of sets could form a hierarchical structure. The element P4 in example 7 is a subroutine which operates on the ANSWER register. For instance, it might be a general indefinite article remover that does the work of examples 1 and 2.

Other set operations might be provided, e.g., intersection, complementation, and element selection.

Note that response sets provide a clean syntactical integration of the natural language processing subsystems of Section 9.5.

9.2.2 The semantics of CASE

The system log showed that students frequently entered PL/I statements in lower case instead of the mandatory upper case. An author would like to identify this error rather than respond with the UNREC message. He can do this in DIAL/1 with the CASE statement, but for this particular need it is clumsy. He would like to put a CASE switch in the body of a sieve and cause the ANSWER register to be retranslated. However, case translation would then no longer be a preprocessing function.

Changing the semantics of CASE thus requires a rethinking of the preprocessor concept in the DIAL machine. Perhaps what is needed is a general function like TRANSLATE in PL/I which would certainly handle the CASE problem.

9.2.3 Block structure

Real author use of DIAL showed certain elements of redundancy. Consider the present typical frame structuring.[3]

```
     SHOWAS
     MATCH response set, branch forward
        .
        .
        .
     MATCH response sets, branch
     MATCH
        .
        .
        .
     UNREC
        .
        .
        .
branchlabel: SHOW feedback
     branch back
        .
        .
        .
```

This shows two redundancies:

(1) the branch to the feedback (juxtaposition would solve it), and

---

[3] See Figure 4.2 for a typical sieve.

(2) the branch from the feedback (always either back to the lead-MATCH or on to the next frame).
The first is a burden not only because of the necessity for generating multitudes of labels, but also because the author must stack feedback messages in his memory as he enters the sieve response sets and then unstack them after the UNREC.

The following vertical bracketing into blocks, one for each sieve element,

```
MATCH
  response sets
  feedback
  branch
MATCH
  .
  .
  .
```

solves the first problem. Also, readability is greatly enhanced. The branch could be dispensed with if the single case of forward branching were distinguished by the operator, MATCHR, so then an implied branch back is part of each ordinary MATCH block.

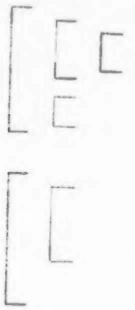
Most labels are obviated. DIAL/1 demands extensive use of labels, an annoying burden in on-line composition. My aim now is self-referencing branches: branches should be relative not absolute.[4]

---

[4] For example, DO-WHILE and UNREC * are self-referencing.

9.2.4 Branching functions

Consider the following block structure



with a set of branching functions which take blocks as (explicit or implicit) arguments:

in	move in to the next contained block
out	move out to the next containing block
up	move to predecessor block on the same level
down	move to successor block on the same level
exit	move to outermost containing block, i.e., *exit* is the function composition of *out's*

Then, viewing a program as a two-dimensional space, horizontal movement comes from the notion of depth into a nesting of blocks. In current DIAL, only vertical motion is possible: amplification of a concept, conceptually a horizontal movement, must be done by a GOTO. Return to a lead match is also done by a GOTO, unless U * is used.

9.2.5 A small number of control structures

DIAL/1 has specialized control structures, e.g., U *. Two new ones are proposed: branching functions and MATCH-block. While specialized control structures in a special purpose language can be justified on the grounds of conciseness of expression and application-orientation, there is the danger that the language may have such a diversity of different control structures that ease of learning and use suffer. Thus, in adding function and providing self-referencing branches, the aim should be a small set of uniform, consistent control structures.

9.2.6 A PARSE system matching function (due to Brooks)

PARSE would take a grammatical specification, in BNF for example, as its argument and return 1 if the student's response parsed correctly, 0 otherwise. For example, if ident names a set of BNF productions for a PL/I identifier, then

PARSE(ident)

could be used in the answer analysis of a response to the question "Construct a valid PL/I identifier."

9.2.7 The operational environment

Whenever DIAL is changed, the implications for the operational environment should be studied. Consider the following.

Indentation enhances the readability of block-structured programs. The)LIST command could reformat an author's statements according to the structure of DIAL as NEATER2 [Conrow and Smith, 1970] does for PL/I. Thus an author need not spend time indenting as he is entering his code.

9.3 Author-defined commands for student use

Currently the only command available to a student is)OFF. All other student inputs are in response to explicit directions from a DIAL program. This section proposes an extension to DIAL which would enable an author to define a set of commands for student users of his instructional program.

Each student response would be checked to see if a command is being given. The occurrence of a command would result in the execution of the action specification for that command. An author would provide a complete definition of each command by a set of command units, each one consisting of a name and action specification. For example,

```
ON REVIEW DO
SHOWAS 'Review sequences are
available for the following
topics.  Indicate your choice
with the light pen:
  * ARITHMETIC OPERATIONS
  *
     .
     .
     .
  * No review - return to lesson'
MATCH PEN(1), ARITH
     .
     .
     .
END
```

A command unit has the form

ON command-name action-specification

where action-specification is a statement or a DO-group.

If execution of the action-specification does not result in a branch out of the command unit, then execution resumes at the SHOW-statement controlling the read.

I suggest that such author-defined commands do not use the special start symbol) since this should be reserved for additional system commands. An author may, however, wish to establish the convention that all begin with some other special symbol, e.g., #, $, or @. This would not be necessary if the rule is established that all MATCH-specified responses are checked before the list of commands.

A command facility would add to the ease of use of DIAL; to achieve the equivalent in the current design, an author must specify the response in every MATCH group, e.g.,

```
MATCH 'REVIEW', REV;
```

Moreover, it would add flexibility to the CAI System; an author could provide for much greater <u>student</u> <u>control</u> over the path through a set of course material.

As a simple example consider a command MENU:

```
ON MENU DO
     SHOWAS'Select the section you wish to
     work next:
          *EXPRESSIONS
          *PROGRAM STRUCTURE
          *DECISION MAKING
          *CHARACTER STRINGS
          *ITERATION
     .
     .
     .
        END
```

The command MENU is roughly equivalent to <u>go</u> <u>to</u> in Coursewriter.

Other examples of author-defined commands are:

(1) a status reporting command, which would display certain measures of performance kept by an author in his DIAL program.

(2) a command by which a student can consult a dictionary or glossary. If the vector DEF contains definitions and is keyed by the vector KEY then an action-specification could be as follows.

```
ON DEFINE DO
   L2: SHOW 'Enter word to be defined'
       ARG <- ANSWER
       I <- 0
   LOOP:  /*Linear search of keys*/
       I  <- I + 1
       IF I > NDEFS THEN GOTO NOT;
              /* not found */
       IF ARG = KEY(I) THEN GOTO FOUND;
       GOTO LOOP;
   FOUND:
       SHOW DEF(I);
       GOTO EXIT; /* return to controlling */
                  /* SHOW via the default  */
   NOT:
       SHOW 'This term is not in the course
            dictionary. Perhaps you misspelled.';
   EXIT:
       END;
```

The responses to Questions IV.2 and IV.3 on the Fall and Spring questionnaires show the desirability of the DEFINE and REVIEW facilities.

Finally, recall the timed-response facility discussed in Section 8.3.2.2 and notice the semantic similarities to the proposed author-defined commands. The action-specification would be

```
ON TIMEUP DO
   .
   .
   .
   .
   .
       END
```

but the <u>system</u> rather than the student would signal the command.

9.4 Debugging aids for DIAL programming

The most important debugging facility is the interactive environment itself. However, this should probably be supplemented by some other aids. Study is needed to determine what aids are necessary and how they can be incorporated into the CAI System in a manner consistent with the design philosophy.

As a departure point for this study, the following could be considered.

1. Tracing

A step-by-step)XEQ option: for every statement executed, two screen displays would appear - a debug screen and a student screen. The debug screen would display information such as

(1) the DIAL statement executed
(2) the values of variables, registers, etc. referenced in the statement.

The author would advance such an execution with the INT key, with successive depressions causing the debug and student screens to appear alternately.

2. Value assignment

This would enable an author to assign values to variables just prior to an)XEQ.

Some aids would be added as part of the language, others as part of the operational environment. Some may be implemented by inserting checks in the object code (subscript checks, for example), others by a monitor controlling author-invoked execution.

Attribute tables, cross-reference lists, etc., should be added to the listing obtained by the)PRINT command.

9.5 Answer processing subsystems

For certain types of constructed responses, e.g., when some meaning must be extracted from a typed response, conventional author languages are sometimes not very powerful. As an alternative to introducing such power into the author language itself, this section proposes a study of existing computer programs for processing semantic information. Suitable programs could be attached as subsystems to the CAI System.

Not only should the systems programming job of attaching subsystems be studied, but also the interface with an author through DIAL.

9.5.1 An existing language processor for the programming language being used in an introductory computer programming course could be harnessed as a subsystem. It would then be possible for an author to elicit a program segment from a student and the CAI System to pass the student's answer to the language processor for analysis and execution. The design challenge here is meaningful communication between the execution and the author's instructional program so that the student can be given pedagogically useful feedback. Project TEACH [Fenichel, 1970] has implemented and evaluated a modest subset of the proposed environment; the project has now launched into a substantial research program aimed at coming closer to the full environment.

As a first step, I suggest a syntax checker. Each program segment would be analyzed syntactically only. A vehicle for this could be the syntax analyzer (using a PL/I grammar) of the CAI TWS described in Chapter 6.

9.5.2 The above subsystems are restricted to only one subject matter. To get real general power, one wants to incorporate natural language processing subsystems. Although Chapter 3 argued that such systems are still too experimental to form the central framework for a production-oriented author-controlled CAI System, system developers must keep this goal in sight. What one really wants for CAI is a subsystem which matches the semantic content of a response against that of a single canonical form given by an author.

9.6 A man-machine interface for unrecognized answers

The following man-machine system is proposed as a generalized unrecognized answer model for research. For each group of n students (n to be determined) there would be a tutor station consisting of a man and a special console. Whenever a student response was not recognized in a DIAL program, the CAI System would route the problem to one of the tutor stations.

The situation is a suitable candidate for a man-machine system. The instructional task would be partitioned such that the inherently algorithmic parts (text display, sequencing, etc.) are handled by the machine and the parts that are best handled by a man (pattern, context and judgment problems) would be allocated to the tutor.

The number of minutes per hour that the tutor spends on the average with each student must be very low for the system to even approach cost-effectiveness.

In terms of the pressure characteristics of the system, the job of tutor would be similar to one in the busiest air traffic control center, where duty periods are as short as one hour.

Some of the challenges involved and questions raised are the following.

1. Information display to the man

The context surrounding the student unrecognized response must be displayed at the console of the tutor station in such a way that the tutor can quickly see the problem and decide on his response.

2. Input from the man

Once the man has decided what his response will be, he must be able to enter it quickly so he can be freed to serve the next problem. Thus typed input is excluded. A menu of responses to be selected by light pen suggests itself. This menu should be part of the problem display. An obvious menu is the set of branch labels appropriate to the problem section of the DIAL program. This, however, requires the man to be intimately familiar with every micro-point in the instructional program. An alternative for the menu might be a "scale of wrongness" which the System would translate into branch labels.

The tutor console might enable the tutor to control a pointer on the student's CRT screen.

3. Multiple interactions

If the tutor cannot decide on an action for a particular problem, he may want to get more information from the student; a two-way channel for multiple interactions per problem is then needed.

4. Instructional programming

There are many implications for the instructional program structure and for the author language.

While the situation proposed may never be implemented, it would serve as a model, the investigation of which should throw light on important problems not only in man-machine dialogue, but also in learning theory.

Finally, note that the system would be ideal for a HELP or HINT facility. When a student types HELP the system would display the context at the tutor station. The tutor could use a voice link for his reply.

9.7 More service programs

Much data is gathered by the CAI System from each student session and stored in various files. Programs over and above the existing ones are needed to analyze this information for authors and proctors. A study should determine

(1) what information is pertinent;

(2) a language for accessing it;

(3) whether batch or interactive programs should be used.

There are three potential sources for information for analysis:

1. The STUREC file

Each student's record contains statistics on terminal time, personal data and the number of recovers and resumes.

2. LOGFILE

This contains, for each student session, student responses (typed or penned) and a trace by DIAL statement number diachronically.

3. The student activation record

This record is called SCB (student control block) in the CAI System. It is not diachronic but a snapshot of the state of all variables, DIAL machine registers, etc., at an instant in time. In contrast to the log file, SCB has instant in time but all in kind. At sign-off it is transferred into SREC.SCB_PART on the student's record in file STUREC.

The work at Florida State University [Davenport, 1968] on the analysis of Coursewriter-generated performance records may be useful in the study.

9.8 Color cathode-ray tube terminals

In the Spring of 1972 the CAI Project added a color option to the display of the work station, a very useful modification. It involved replacing the existing CC-300 TV Display by a color TV and modifying the CC-301 TV Display Controller. Characters can be displayed in green, red, blue or yellow.

Character color is specified by four (non-displayed) control characters. These codes are:

(1) entered from the keyboard by simultaneous depression of the special code key (SP) and one of Q,R,S,or T, or

(2) sent from the computer.

Once a color selection has been made, all characters received by the controller are stored with that color designation until the next color code is received. Thus color codes act as shift characters. But when a message is transmitted from the CC-301 to the computer these control characters cannot be retrieved. Because of this hardware deficiency, software is needed to transmit an encoding of these characters to the computer so that when messages are sent back the appropriate color control characters will be inserted.

An interim solution has been implemented. Four system text constants, GREEN, RED, BLUE, and YELLOW, were added to the language. They can appear in a DIAL text-expression just as any other text constant. For example,

```
SHOW 'The following is an example of an  '
   ||RED||'arithmetic ' ||BLUE|| 'expression' .
```

A better solution, in that it does not intrude on the language, is to make color changing part of the operational environment. Four light buttons (one for each color) at the bottom of the author mode display format could make color changing appear to be in the hardware. Thus, to show "arithmetic" in red, as above, an author would enter

```
SHOW 'The following is an example of an
        arithmetic expression.'
```

but after "an" he would pen the red light button. The cursor would then change color; after "arithmetic" he would pen blue.

This solution is perhaps as neat as can be done by software alone. The <u>right</u> solution from a human factors viewpoint requires minor hardware changes.

APPENDIXES

APPENDIX A

QUESTIONNAIRE A AND SUMMARIES OF STUDENT RESPONSES

For each question, four sets of responses are given. They represent the totals from each class, as shown in the following example.

6	8	2		←	Fall, 1972, Conventional method-Instructor 1
1	15	4	1	←	Fall, 1972, Conventional method-Instructor 2
2	12	7		←	Fall, 1972, CAI
8	30	17	1	←	Spring, 1973, CAI

In the Fall, 58 students out of a possible 60 completed the questionnaire. In the Spring, 56 out of a possible 69 completed it.

NAME:

COMPUTER-ASSISTED INSTRUCTION STUDY

FALL 1972

Student questionnaire for control groups and experimental group

Please fill out the attached questionnaire as completely and accurately as possible. The information gathered will help planning for more effective methods of teaching computer science courses.

You will not be graded on your responses to this questionnaire. In fact, neither your instructor nor the course supervisor will ever see any completed questionnaire. Neither will any student be listed or mentioned by name or number in reports describing the results of this study.

To be completed in class on Monday, October 2, 1972.

Mudge
J. C. Mudge
Experimenter

JCM/vj
9 72

1. Year at UNC:

Fresh-man	Soph-omore	Junior	Senior	Grad-uate Student	Other (Please specify ________)
2	3	4	1	5	1
2	4	5	10		
	2	1	9	5	4
6	*8*	*14*	*13*	*12*	*3*

2. Major or intended major:

____

3. How would you rate your attitude to the subject matter of this course? Restrict your rating to the subject matter itself, not the method or quality of instruction.

very positive	positive	neutral	negative	very negative
9	4	3		
2	14	5		
9	6	5	1	
14	*34*	*7*	*1*	

4. What previous experience have you had with computers?

Circle all that apply:

8	12	10	*37*:	none
		1	*5*:	other programming courses
	1		*1*:	computer appreciation course
	1	3	:	this course before
6	6	5	*13*:	use of pre-programmed packages, e.g., statistical packages
1	3	2	*3*:	programming in language(s) other than PL/I
1	2	5	*10*:	other (Please specify)

5. Why are you taking this course?

Circle all that apply:

1	1	2	*6*:	it is a prerequisite for other courses I plan to take
2			*1*:	it is required in my program
4		2	*9*:	it satisfies the language requirement for a graduate degree
6	10	9	*34*:	solely for a broadening experience
4	3	8	*11*:	it will help me get a job
2	5		*6*:	it satisfies the math requirement for my degree
10	12	15	*41*:	programming will be a useful research skill
3	6	2	*7*:	other (please specify)

6. (a) At the beginning of the semester I felt that obtaining a good grade in the course was

4	10	2
13	8	
6	12	3
22	*25*	*9*
very important	somewhat important	not important

(b) At this time, I feel that obtaining a good grade in the course is

3	11	2
13	8	
7	9	5
21	*25*	*10*
very important	somewhat important	not important

4

7. (a) In comparison with courses which have a similar relationship (e.g., elective, required course) to my program, the amount of time I now allot to this course is

much more than average	more than average	average	less than average	much less than average
5	9	2		
6	12	2	1	
13	6	1		
9	*28*	*14*	*2*	*1*

(b) At the beginning of the course I had expected this time allocation to be

much more than average	more than average	average	less than average	much less than average
6	8	2		
1	15	4	1	
2	12	7		
8	*30*	*17*	*1*	

8. This course has been frustrating:

all of the time	most of the time	some of the time	only occasionally	never
	1	7	6	2
1	5	10	5	
3	4	12	2	
3	*5*	*28*	*15*	

9. (a) With respect to intellectual challenge, before I began the course I expected the subject matter to be

very challenging	challenging	about average	trivial	very trivial
3	8	5		
2	13	5	1	
2	11	8		
6	37	13		

(b) With respect to intellectual challenge, I would now rate the subject matter as

very challenging	challenging	about average	trivial	very trivial
2	4	9	1	
5	11	5		
7	11	2	1	
10	20	16	1	

10. (a) Before I began the course I would have rated the subject matter as

very valuable	valuable	about average	worthless	very worthless
5	7	3	1	
2	17	2		
8	10	3		
18	35	9		

10. (b) I now rate the subject matter as

5	7	3	1	
2	17	1	1	
6	9	6		
9	36	12		
very valuable	valuable	about average	worthless	very worthless

11. Do you believe that teaching can be automated?

12	4
10	11
15	6
36	17
yes	no

APPENDIX B

QUESTIONNAIRE B AND SUMMARIES OF STUDENT RESPONSES

For each question, two sets of responses are given. The italicized set represents the total from the Spring class, the other set represents the Fall class. Some questions, e.g., Section III, Questions 10 and 11, were not applicable to the Spring class and so were not printed on the Spring questionnaire.

Section II of this questionnaire is a "student reaction inventory"[1] developed at Pennsylvania State University.

In the Fall, 21 students out of a possible 22 completed the questionnaire. In the Spring, 56 out of 69 completed Sections I to III, and 36 out of 69 completed Section IV.

---

[1] The Development and Presentation of Four College Courses by Computer Teleprocessing. Computer Assisted Instruction Laboratory, The Pennsylvania State University, University Park, Pennsylvania. June 30, 1967.

NAME:

COMPUTER-ASSISTED INSTRUCTION STUDY

FALL 1972

Student Questionnaire for the CAI Group

Please fill out the attached questionnaire as completely and accurately as possible. The information gathered will be used to enhance the Department of Computer Science's CAI System. We are seeking information, not compliments; please be frank.

You will not be graded on your responses to this questionnaire. Neither will any student be listed or mentioned by name or number in reports describing the results of this study.

J. C. Mudge
Experimenter

Sections I, II and III are to be completed in class on Monday, October 2, 1972. Section IV is to be handed in at the beginning of class on Wednesday, October 4, 1972.

JCM/vj

SECTION I

1. I have had contact with computer-assisted instruction prior to this course:

20	1
51	5
no	yes (please specify)

2. My initial reaction when informed that the first weeks of instruction would be by CAI was

12	6	3		
9	30	12	5	
very favorable	favorable	indifferent	unfavorable	very unfavorable

3. I enjoyed the scheduling freedom provided by CAI

			6	15
1		4	21	30
strongly disagree	disagree	uncertain	agree	strongly agree

4. I use a typewriter keyboard

7	7	4	3	
12	*32*	*8*	*2*	*2*
by touch fluently	by touch haltingly	hunt-and-peck rapidly	hunt-and-peck haltingly	(not familiar with a typewriter keyboard)

5. I prefer the lesson units to be

3	15	2
9	*44*	*3*
longer	the length they were	shorter

6. My first and second choices for lesson length are [Responses were weighted for - 2 for first choice, 1 for second.]

8	11	11	13	8
2	*16*	*41*	*54*	*48*
10 mins	20 mins	30 mins	45 mins	60 mins

7. I would like more opportunity during a CAI session to review

2	15	1	3
2	*43*	*3*	*8*
complete lessons	parts of lessons, both slides and questions	slides only	(no review)

8. I reviewed slide handouts between CAI sessions

3	8	5	3	2
12	*15*	*16*	*11*	*2*
all the time	most of the time	some of the time	only occasionally	never

9. After each CAI session I would like a printed copy of the questions and answers covered in the session

		1	5	15
1		*4*	*24*	*27*
strongly disagree	disagree	uncertain	agree	strongly agree

10. For entering my answer to a multiple-choice question, I prefer to use

5	16
19	*34*
the light pen	numbers entered from the keyboard

11. Messages on the TV screen were removed too quickly

	3	6	11	1
		10	*25*	*21*
all the time	most of the time	some of the time	only occasionally	never

12. I prefer to press INT to signal when I have read a message on the TV screen

		1	12	8
		4	32	20
strongly disagree	disagree	uncertain	agree	strongly agree

13. Some slides had no questions. I prefer one or more questions after each slide

2	7	6	5	1
1	22	18	12	3
strongly disagree	disagree	uncertain	agree	strongly agree

14. I found the weekly question-and-answer sessions with the instructor to be

	7	11	3
5	30	18	3
very helpful	sometimes helpful	not very helpful	(I didn't go)

15. Assuming three contact hours per week, what would be the best balance between CAI and in-class instruction?

	6	12	1	2
	10	*39*	*4*	*2*
CAI hours	3	2	1	0
in-class hours	0	1	2	3

16. Compared to a regular in-class lecture, during a CAI session I felt I had to concentrate

5	10	4	2	
11	*30*	*9*	*6*	
much more	more	about the same	less	much less

17. When I entered an answer the computer responded with adequate speed

4	16	1		
5	*38*	9	*4*	
all the time	most of the time	some of the time	only occasionally	never

SECTION II

CIRCLE THE RESPONSE THAT MOST NEARLY REPRESENTS YOUR REACTION TO EACH OF THE STATEMENTS BELOW:

1. While taking Computer Assisted Instruction I felt challenged to do my best work,

	3	4	9	5
2	14	13	21	6
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

2. The material presented to me by Computer Assisted Instruction caused me to feel that no one really cared whether I learned or not.

7	12		1	1
12	34	6	2	1
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

3. The method by which I was told whether I had given a right or wrong answer became monotonous.

2	7	3	9	
3	24	9	12	8
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

4. I was concerned that I might not be understanding the material.

1	5	2	12	1
1	22	6	23	4
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

5. I was not concerned when I missed a question because no one was watching me anyway.

7	9		5	
9	28	6	13	
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

6. While taking Computer Assisted Instruction I felt isolated and alone.

2	3		3	13
3	4	4	16	29
All the time	Most of the time	Some of the time	Only occasionally	Never

7. While taking Computer Assisted Instruction I felt as if someone were engaged in conversation with me.

1	1	5	6	8
4	7	17	10	18
All the time	Most of the time	Some of the time	Only occasionally	Never

8. The responses to my answers seemed appropriate.

All the time	Most of the time	Some of the time	Only occasionally	Never
	14	6	1	
2	26	22	6	1

9. I felt uncertain as to my performance in the programmed course relative to the performance of others.

All the time	Most of the time	Some of the time	Only occasionally	Never
3	3	5	7	3
4	7	12	17	15

10. I found myself just trying to get through the material rather than trying to learn.

All the time	Most of the time	Some of the time	Only occasionally	Never
1	1	5	7	7
2	2	18	21	13

11. I knew whether my answer was correct or not before I was told.

Quite often	Often	Occasionally	Seldom	Very Seldom
4	7	6	3	1
3	31	15	5	2

12. I guessed at the answers to questions.

Quite often	Often	Occasionally	Seldom	Very Seldom
1	1	11	2	6
2	2	30	16	6

13. In a situation where I am trying to learn something, it is important to me to know where I stand relative to others.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
3	11	1	4	2
6	25	13	11	1

14. I was encouraged by the responses given to my answers of questions.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
1	3	7	9	1
2	14	17	23	

15. As a result of having studied some material by Computer Assisted Instruction, I am interested in trying to find out more about the subject matter.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
1	2	5	11	2
	6	15	34	1

16. In view of the time allowed for learning, I felt too much material was presented.

All the time	Most of the time	Some of the time	Only occasionally	Never
	1	5	8	7
	2	4	14	36

17. I was more involved in running the machine than in understanding the material.

All the time	Most of the time	Some of the time	Only occasionally	Never
		2	7	12
1	*2*	*5*	*31*	*17*

18. I felt I could work at my own pace with Computer Assisted Instruction.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
	1	2	9	9
	3	*1*	*32*	*20*

19. Computer Assisted Instruction makes the learning too mechanical.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
5	11	2	2	1
13	*32*	*6*	*5*	

20. I felt as if I had a private tutor while on Computer Assisted Instruction.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
1	7	2	10	1
1	*21*	*11*	*16*	*7*

21. I was aware of efforts to suit the material specifically to me.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
1	5	9	6	
6	*24*	*16*	*9*	*1*

22. I found it difficult to concentrate on the course material because of the hardware.

All the time	Most of the time	Some of the time	Only occasionally	Never
			11	10
		4	*24*	*28*

23. The Computer Assisted Instruction situation made me feel quite tense.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
7	11	2		1
18	*33*	*2*	*3*	

24. Questions were asked which I felt were not relevant to the material presented.

All the time	Most of the time	Some of the time	Only occasionally	Never
		1	10	10
	1	*4*	*30*	*21*

25. Computer Assisted Instruction is an inefficient use of the student's time.

Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
8	10	3		
21	*25*	*5*	*4*	*1*

26. I put in answers knowing they were wrong in order to get information from the machine.

Quite often	Often	Occasionally	Seldom	Very Seldom
1	1	8	4	7
1	*6*	*21*	*15*	*13*

27. Concerning the course material I took by Computer Assisted Instruction, my feeling toward the material before I came to Computer Assisted Instruction was:

Very favorable	Favorable	Indifferent	Unfavorable	Very Unfavorable
3	11	5	2	
5	*33*	*17*		

28. Concerning the course material I took by Computer Assisted Instruction, my feeling toward the material after I had been on Computer Assisted Instruction is:

Very favorable	Favorable	Indifferent	Unfavorable	Very Unfavorable
2	17	2		
8	*37*	*9*		

29. I was given answers but still did not understand the questions.

Quite often	Often	Occasionally	Seldom	Very Seldom
1		11	3	6
1	*3*	*28*	*14*	*10*

30. While on Computer Assisted Instruction I encountered mechanical malfunctions.

Quite often	Often	Occasionally	Seldom	Very Seldom
	1	10	6	4
2	*6*	*24*	*12*	*11*

31. Computer Assisted Instruction made it possible for me to learn quickly.

		5	12	3
1	5	21	24	5
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

32. I felt frustrated by the Computer Assisted Instruction situation.

3	12	3	2	1
11	31	6	6	1
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

33. The responses to my answers seemed to take into account the difficulty of the question.

	7	6	8	
1	19	17	17	1
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

34. I could have learned more if I hadn't felt pushed.

4	8	6	2	1
6	35	7	6	
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

35. The Computer Assisted Instruction approach is inflexible.

1	9	4	6	1
1	38	10	5	1
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

36. Even otherwise interesting material would be boring when presented by Computer Assisted Instruction.

4	14		2	1
8	37	6	4	
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

37. In view of the effort I put into it, I was satisfied with what I learned while taking Computer Assisted Instruction.

1		3	13	4
2	6	5	36	6
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

38. In view of the amount I learned, I would say Computer Assisted Instruction is superior to traditional instruction.

2	1	7	7	4
2	13	18	17	4
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

39. With a course such as I took by Computer Assisted Instruction, I would prefer Computer Assisted Instruction to traditional instruction.

1	3	2	10	5
2	7	17	21	8
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

40. I am not in favor of Computer Assisted Instruction because it is just another step toward de-personalized instruction.

5	14		1	1
10	29	8	7	2
Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree

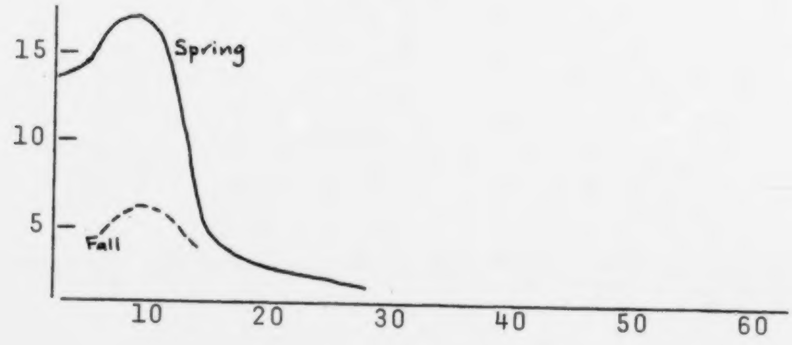
SECTION III

1. The West House CAI Center was a smoothly running operation.

	15	6		
7	41	6	1	
all the time	most of the time	some of the time	only occasionally	never

2. How long did it take you to get used to the operation of the CAI System?

_________ minutes



3. Did you receive adequate instruction in operating the work station?

15	6
53	2
yes	no (please suggest improvements)

4. How many scheduled CAI sessions did you miss because the CAI System was inoperative?

0	8 *43*
1	9 *17*
2	4 *3*
3	1 *4*
4	*1*

5. How many scheduled CAI sessions did you miss because the next batch of course material was not ready?

0	12
1	7
2	2
3	1

6. During my CAI sessions I requested help from the proctor

1 *1*	8 *15*	11 *32*	1 *4*	*3*
quite often	often	occasionally	seldom	very seldom

7. I consider the availability of a proctor to be

17	3	1	
41	11	3	
essential	highly desirable	desirable	not essential

8. I consider the proctor call switch to be a good method of summoning the proctor.

17	4
54	1
yes	no (Please give reason)

9. When I needed assistance from a proctor I felt he was familiar with Dr. Brooks's course material.

7	11	1	2	
27	19	4	2	2
all the time	most of the time	some of the time	only occasionally	never

10. For experimental control reasons, the proctor was not allowed to answer subject matter questions.
If this restriction was removed I would prefer to be able to ask the proctor questions on the subject matter.

	1	3	12	5
strongly disagree	disagree	uncertain	agree	strongly agree

11. I had difficulty reading the black-and-white slides

	1	4	8	8
all the time	most of the time	some of the time	only occasionally	never

12. Viewing the TV screen and slide screen resulted in eyestrain

			8	13
	2	*4*	*12*	*37*
all the time	most of the time	some of the time	only occasionally	never

13. I found the work station room to be
(Circle all that apply)

2				18
6	*21*	*12*	*2*	*29*
too cold	too hot	stuffy	too noisy	comfortable

14. I prefer the room lighting to be

9	12
43	*8*
on	off

15. I prefer to work with the work station door

3	18
10	*44*
open	closed

16. I prefer the keyboard height to be

	20	1
1	52	2
lower	about the same	higher

17. I was distracted by noise outside the work station

	6	15
	7	48
often	sometimes	never

18. I felt uncertain as to how I should be making use of the prescribed texts for this course

1	1	1	14	4
2	13	5	26	9
strongly disagree	disagree	uncertain	agree	strongly agree

NAME:

Student Questionnaire for the CAI Group

SECTION IV

(To be handed in at the beginning of class on Wednesday, October 4, 1972)

1. A hint facility in a CAI system would respond with an author-prepared hint when requested during a question/answer sequence.

   I view such a hint facility as

6	11	4	
4	18	12	2
mandatory	highly desirable	desirable	not necessary

2. A definition facility in a CAI system would allow a student to consult an author-prepared dictionary or glossary stored in the system. Sample student requests might be

```
)define     variable

)define     ||

)define     INDEX
```

   I view such a definition facility as

3	10	8	
7	21	7	1
mandatory	highly desirable	desirable	not necessary

3. A student-controlled review facility in a CAI system would enable a student to request a review at any time in a session. The request)review would result in the display of a menu of topics available for review. The student would light pen his selection.

In the CAI course you have just taken, opportunities for review were given in later lessons and then only at author-specified points.

I view a student-controlled review facility as

9	7	4	1
7	18	9	2
mandatory	highly desirable	desirable	not necessary

4. I would like all of the courses this semester to be on the CAI system

3	6	5	5	2
9	12	8	4	2
strongly disagree	disagree	uncertain	agree	strongly agree

[This question is of doubtful use because of a typographical error - I intended "courses" to be singular.]

5. I suggest the following improvements to the work station:

6. I suggest the following improvements to the West House operation:

7. Of the courses I have taken I feel that the following are suited to presentation by CAI:

8. Any other comments:

JCM/vj
1072

APPENDIX C

POSTTEST

(5 min.) 1. What is an algorithm? Give a brief example of an algorithm.

(10 min.) 2. Distinguish between variables and values. How may variables be given values?

(5 min.) 3. Find all of the errors in each of the PL/C statements below. Use the space provided beneath each statement to describe the errors in the statement. If a statement contains no errors, indicate this by writing NO ERRORS FOUND beneath the statement.

```
LABEL 1: GET LIST X,Y,Z;


A + B = C;


X = 24X + Y;
```

(10 min.) 4. Two numbers are in variables FIRST and SECOND. Write a PL/C statement using IF to put the larger number into a variable called BIG. Assume everything has been properly declared.

(20 min.) 5. What is the output from this program? Draw a box around your answer.

<u>Hint</u>: Trace through the program as if it were executing. Note each change in the value of any variable. If your answer is incorrect, any partial credit given will be based on your trace. If necessary, you may use the blank page following this one.

```
 PGM: PROC OPTIONS (MAIN);
 DCL (I,Y) FIXED;
 DODO: DO I = 1 TO 5;
       GET LIST (Y);
       IF Y <  5 THEN L: DO;
                         PUT LIST (Y);
                         Y = 2 * Y;
                         END L;
       PUT LIST (Y);
       END DODO;
 END PGM;
*DATA
 1, 2, 4, 8, 16
```

SELECTED BIBLIOGRAPHY

SELECTED BIBLIOGRAPHY

Alpert, D., and Bitzer, D. L. Advances in computer-based education. Science 167, (March 20, 1970), 1582-1590.

Avner, R. A., and Tenczar, Paul. The TUTOR manual. Computer-based Education Research Laboratory, University of Illinois - Urbana, 1969.

Bitzer, Donald, and Skaperdas, D. The economics of a large-scale computer-based education system: Plato IV. In Computer-assisted Instruction, Testing, and Guidance, Wayne H. Holtzman, (Ed.), Harper and Row, New York, 1970, 17-29.

Brooks, F. P. Jr., Ferrell, J. K., and Gallie, T. M. Organizational, financial, and political apsects of a three-university computing center, Proc. IFIP Congress 1968, North Holland Publishing Co., Amsterdam, 923-927.

Brooks, F. P. Jr. Computer-man communication: Using computer graphics in the instructional process. In Advances in Computers, W. Freiberger, (Ed.), Academic Press, New York, 1970, 149-173.

Bunderson, C. Victor. The computer and instructional design. In Computer-assisted Instruction, Testing, and Guidance, Wayne H. Holtzman, (Ed.), Harper and Row, New York, 1970, 45-73.

Bundy, Robert F. Computer-assisted instruction -- Where are we? Phi Delta Kappan 49, 8 (Apr. 1968), 424-429.

Carbonnel, J. R. AI in CAI: An artificial intelligence approach to computer-assisted instruction. IEEE Transactions on Man-Machine Systems, MMS-11, 4 (Dec. 1970), 190-202.

Cheatham, T. E. The Theory and Construction of Compilers, Excerpts, Second Edition. Computer Associates, Wakefield, Mass. 1967.

Computer Communications, Inc. CC-30 Communications Station Reference Manual. 1968.

Conrow, K., and Smith, R. G. NEATER2: A PL/I source statement reformatter. Communications of the ACM 13, 11 (Nov. 1970), 669-675.

Davenport, B. A. and others. The Florida State University Data Management System for the IBM 1500/1800 Instructional System (Introductory Documentation). Computer-Assisted Instruction Center, Florida State University, July, 1968.

Dearborn, R. O. A study of the value of hard copy output in computer-assisted instruction. M.S. Thesis, University of North Carolina, Chapel Hill, 1970.

Dijkstra, E. W. Notes on Structured Programming. EWD 249, Technical Report, Technical University Eindhoven, 1970.

Feingold, S. L. PLANIT - A flexible language designed for computer-human interaction. Proc. AFIPS 1967 SJCC, Vol. 30, AFIPS Press, Montvale, N. J., pp. 545-552.

Feldman, J. A., and Gries, D. Translator writing systems. Communications of the ACM 11, 2 (Feb. 1968), 77-113.

Fenichel, R. R., Weizenbaum, J., and Yochelson, J. C. A program to teach programming. Communications of the ACM 13, 3 (Mar. 1970), 141-146.

Freeman, D. N. A storage hierarchy system for batch processing. Proc. AFIPS 1968 SJCC, Vol. 32, AFIPS Press, Montvale, N. J., pp. 229-243.

Freeman, D. N., and Pearson, R. R. Efficiency vs responsiveness in a multiple-services computer facility. Proc. ACM, 23rd Nat. Conf., 1968, 25-34b.

Gries, D. Compiler Construction for Digital Computers. Wiley, New York, 1971.

Hansen, D. N. Development of CAI Curriculum. In *CAI Center, Florida State University: Annual Progress Report, January 1, 1968, through December 31, 1968*, Florida State University, Tallahassee, Florida, 1969, 119-138.

Hesselbart, J. C. FOIL - a file-oriented interpretive language. *Proc. ACM*, *23rd Nat. Conf.*, 1968, 93-98.

Hesselbart, J. C., D'Arms, T., and Zinn, K. L. *File-oriented Interpretive Language*, Part I, A Manual for Authors. University of Michigan, April, 1969.

HumRRO. *Project IMPACT - Computer-Administered Instruction: Description of the Hardware/Software Subsystem.* Human Resources Research Organization, December, 1970.

Hunt, E., and Zosel, M. Writeacourse: An educational programming language. *Proc. AFIPS 1968 FJCC*, Vol. 33, AFIPS Press, Montvale, N. J., pp. 923-928.

IBM Corporation. *IBM 1500 Coursewriter II Author's Guide*, Form Y26-5681. July, 1968.

________. *Coursewriter III for System/360-Version 2, Application Description Manual*, Form H20-0587. August, 1969.

________. *Coursewriter III for System/360-Version 2, Author's Guide*, Form GH20-0609. December, 1969.

________. *Coursewriter III for System/360-Version 2, Student/Monitor Users Guide*, Form GH20-0608. September, 1969.

________. *Coursewriter III for System/360-Version 2, Supervisor's Guide, Form GH20-0610*. January, 1970.

________. *IBM System/360 Operating System, PL/I (F) Language Reference Manual*, Form GC28-8201. June, 1970.

Kirk, R. E. *Experimental Design Procedures for the Behavioral Sciences*. Brooks/Cole, Belmont, 1968.

McKeeman, W. M., Horning, J. J., and Wortman, D. B. *A Compiler Generator*. Prentice-Hall, Englewood Cliffs, 1970.

Meadow, C. T., Waugh, D. W., and Miller, F. E. CG-1, a course generating program for computer-assisted instruction. Proc. ACM, 23rd Nat. Conf., 1968, 99-110.

Mudge, J. C. UNC CAI System - Systems Programmer Manual, Department of Computer Science, University of North Carolina, Chapel Hill, 1972.

Naur, P. (Ed.) Revised report on the algorithmic language ALGOL 60. Communications of the ACM 6, 1 (Jan. 1963), 1-17.

Newman, W. M. A system for interactive graphical programming. Proc. AFIPS 1968 SJCC, Vol. 32, AFIPS Press, Montvale, N. J., pp. 47-54.

Oettinger, A. G. Run, Computer, Run. Collier, New York, 1969.

Oldehoeft, A. E. Analysis of constructed mathematical responses by numeric tests for equivalence. Proc. ACM, 24th Nat. Conf., 1969, 117-124.

Oliver, P., and Brooks, F. P., Jr. Evaluation of an interactive display system for teaching numerical analysis. Proc. AFIPS 1969 FJCC, Vol. 35, AFIPS Press, Montvale, N. J., pp. 525-533.

Pakin, S. APL/360 Reference Manual, Science Research Associates, Inc., Chicago, 1968.

Radin, G., and Rogaway, H. P. Highlights of a new programming language. Communications of the ACM 8, 1 (Jan. 1965), 9-17.

Sackman, H. Computers, System Science, and Evolving Society. Wiley, New York, 1967.

Sackman, H. Man-Computer Problem Solving. Auerbach, Princeton, 1970.

Sammet, J. E. Programming Languages: History and Fundamentals. Prentice-Hall, Englewood Cliffs, 1969.

Schultz, G. D. CHAT: An OS/360 MVT time-sharing subsystem for displays and teletype. M.S. Thesis, University of North Carolina, Chapel Hill, 1973.

Simmons, R. F. Natural language question-answering systems: 1969. Communications of the ACM 13, 1 (Jan. 1970), 15-30.

Stolurow, L. M., and Peterson, T. I. Harvard University Computer-Aided Instruction (CAI) Laboratory, Technical Report No. 6, March, 1968.

Suppes, P., Jerman, M., and Brian, D. Computer-Assisted Instruction: Stanford's 1965-66 Arithmetic Program. Academic Press, New York, 1968.

Suppes, P., and Morningstar, M. Computer-assisted instruction. Science 166, (Oct. 17, 1969), 343-350.

Thompson, F. B., Lockeman, P. C., Dostert, B., and Deverill, R. S. REL: A rapidly extensible language system. Proc. ACM, 24th Nat. Conf., 1969, 399-417.

Walters, J. Private communication. 1970.

Weizenbaum, J. ELIZA - a computer program for the study of natural language communications between man and machine. Communications of the ACM 9, 1 (Jan. 1966), 36-45.

Wexler, J. D. A generative, remedial and query system for teaching by computer. Ph.D. Thesis, University of Wisconsin, 1970.

Wexler, J. D. Information networks in generative computer-assisted instruction. IEEE Transactions on Man-Machine Systems, MMS-11, 4 (Dec. 1970), 181-189.

Zinn, K. L. A comparative study of languages for programming interactive use of computers in instruction. EDUCOM, Boston, 1969.